Machine Learning-based Analysis of Nanoindentation Test Results on Tantalum

Abhay Komanduri, Yutaro Sugimoto,
Sayem Bin Abdullah, Moses P Varghese

MSE 542

Outline

- Background of the Study
- Introduction to the test data sets
- Types of nano-indentation methods
- Correlation among Rapid Test and CSM Test
- Prediction of Berkovich Hardness in As-built Condition
- Unsupervised Machine Learning Approach in Categorizing the data
- Conclusion

Background and Dataset

Data Title: Quantifying heterogeneous deformation in grain boundary regions on shock loaded tantalum using spherical and sharp tip nanoindentation

- Study the nanoindentation behavior of additively manufactured (AM) Tantalum (Atomic Number 73 in periodic table)

Tantalum

Additive manufacturing

- Two Post-Processing Condition: Shock-loaded and As-built condition
- Data sets of nano-indentation in shock-loaded three grains along their boundary
- Data set included: Raw data of nanoindentation and EBSD map

Weaver, J. S., Jones, D. R., Li, N., Mara, N., Fensin, S., & Gray, G. T. (2018). Quantifying heterogeneous deformation in grain boundary regions on shock loaded tantalum using spherical and sharp tip nanoindentation. *Materials Science and Engineering: A*, *737*, 373–382. https://doi.org/10.1016/J.MSEA.2018.09.075

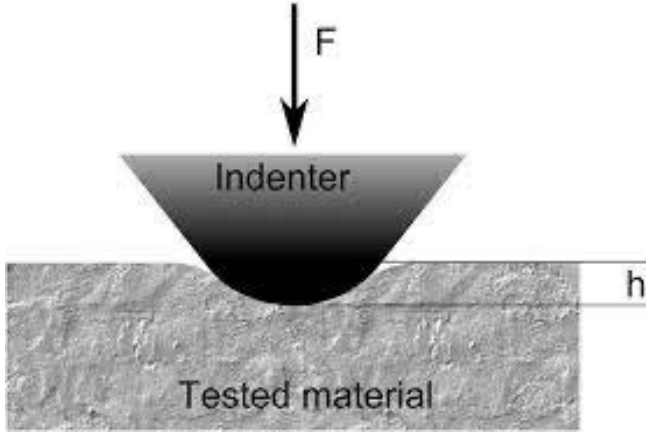
Nano-indentation

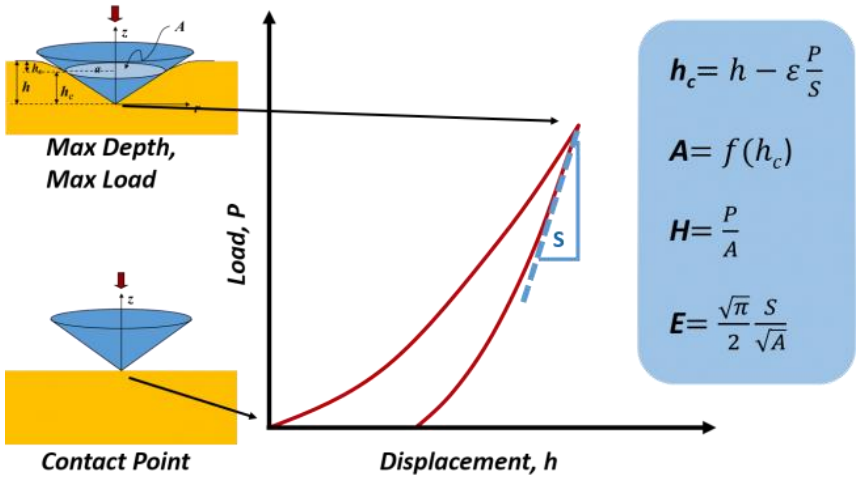
1. Nano-indentation is powerful test method to measure hardness (resistance to deformation), yield strength and modulus at grain level (nanometer level)

2. The hardness and modulus are measured from the raw load displacement curve during unloading

3. Two types of nano indentation methods based on indenter geometry (Berkovich and Spherical)

4. Two types of berkovich test method- a) Rapid Test, b) CSM test





$$h_c = h - \varepsilon \frac{P}{S}$$

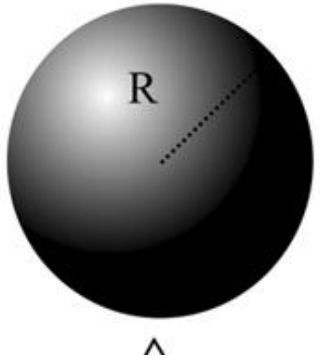
$$A = f(h_c)$$

$$H = \frac{P}{A}$$

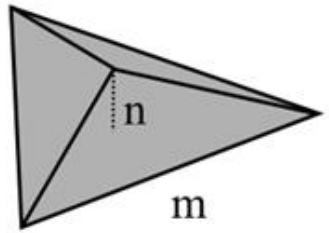
$$E = \frac{\sqrt{\pi}}{2} \frac{S}{\sqrt{A}}$$

Spherical Shape

R=400 μm



Berkovich Shape

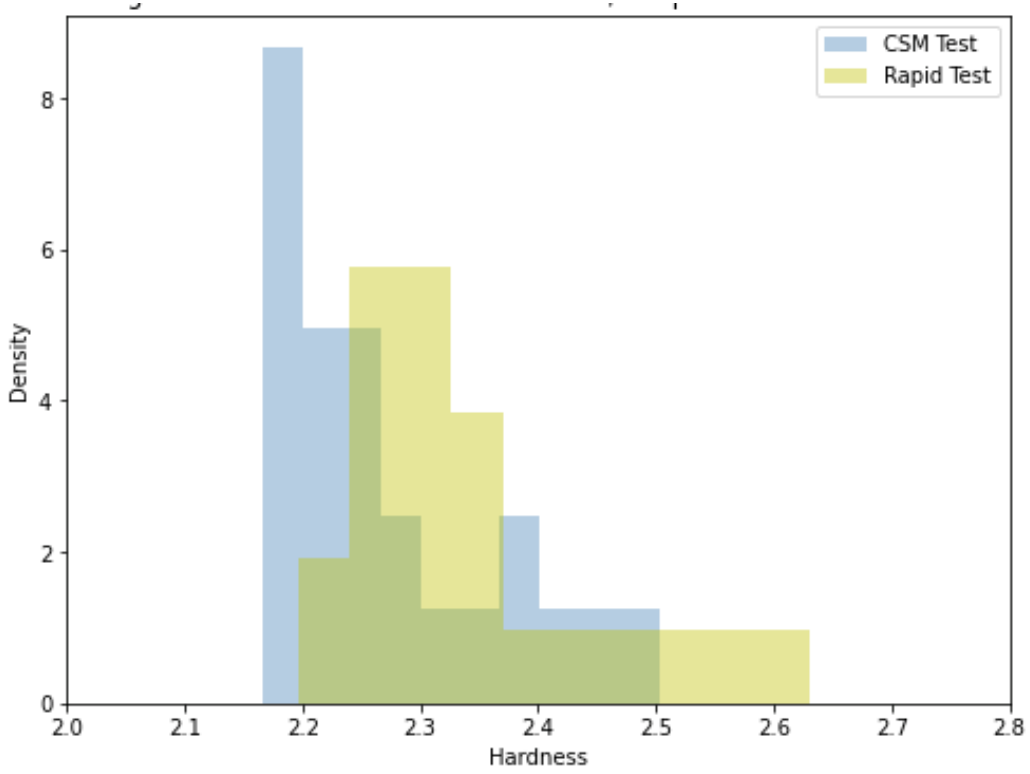


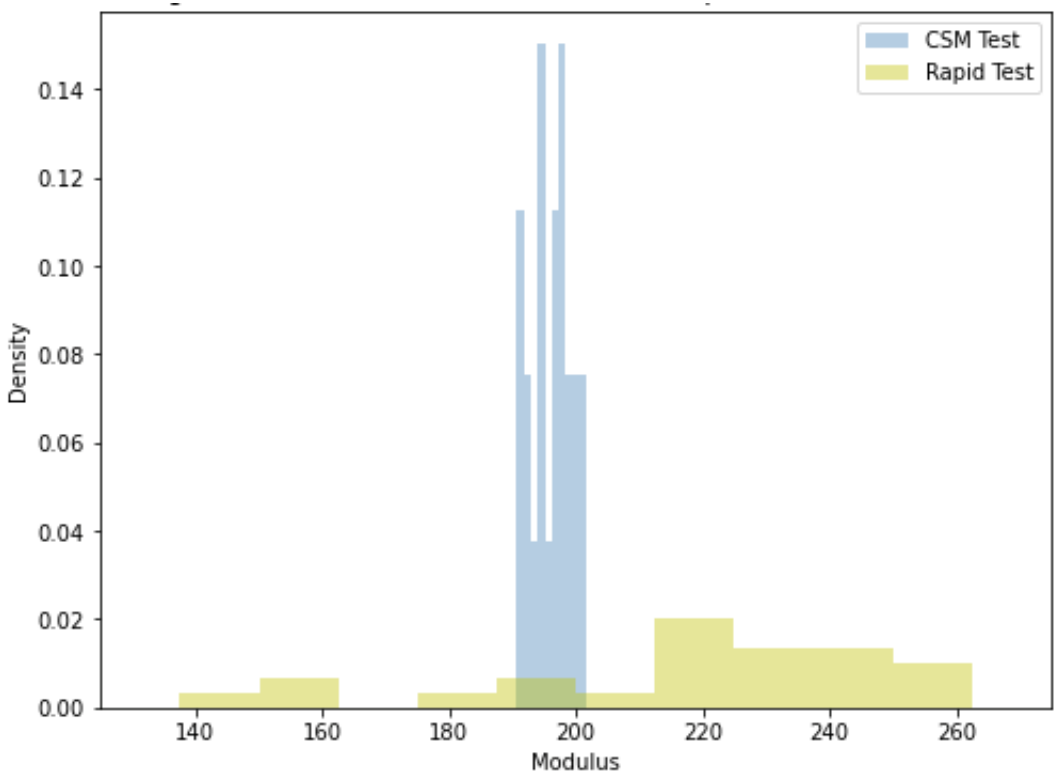
n/m=0.133

Objective of the study

- Study the relationship between Two types of Berkovich test (Rapid test and Contineous Stiffness Measurement Test) using machine learning algorithm
- Prediction of nanohardness properties using machine learning algorithm
- Application of unsupervised machined learning in nanoindentation data
- Challenges in the Dataset and Recommendations

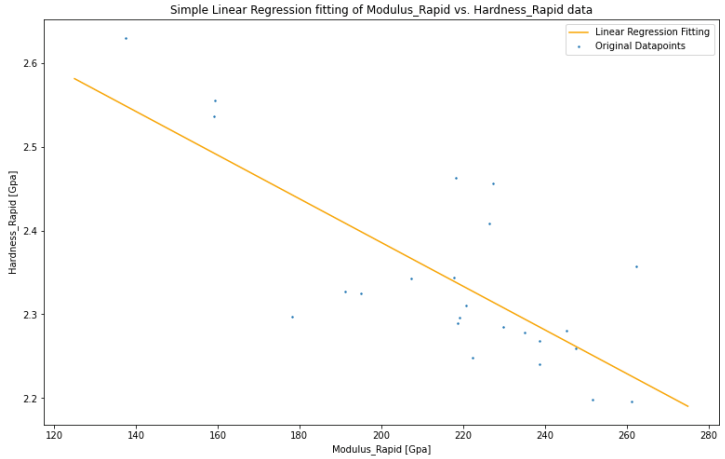
Hardness and Modulus – CSM and Rapid test

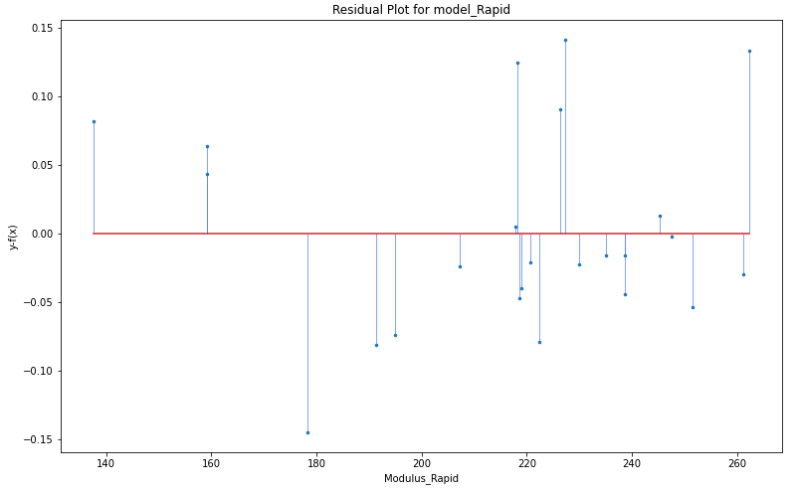




Linear regression – hardness vs. modulus (Rapid and CSM)

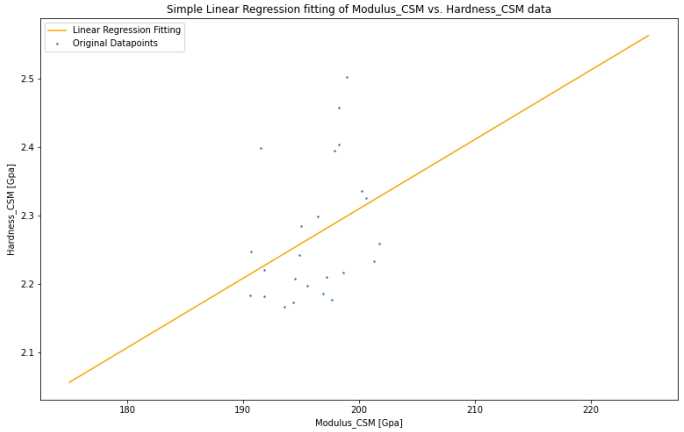
Rapid Test

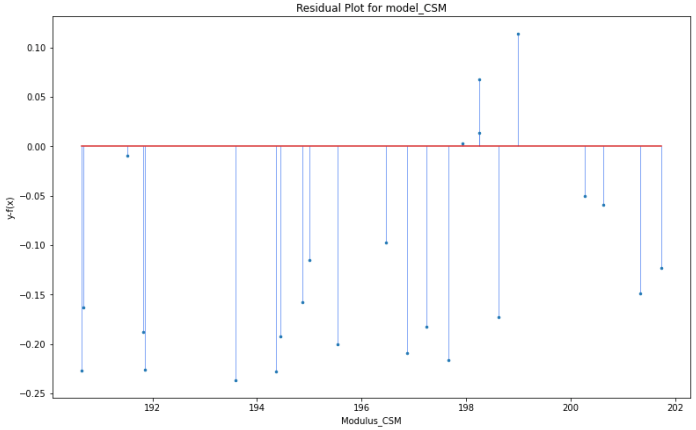




R^2 SCORE = 0.5683983099345535
beta_0, beta_i: 2.907008276184499 [-0.0026065]
Mean Squared Error = 0.005232607618535584

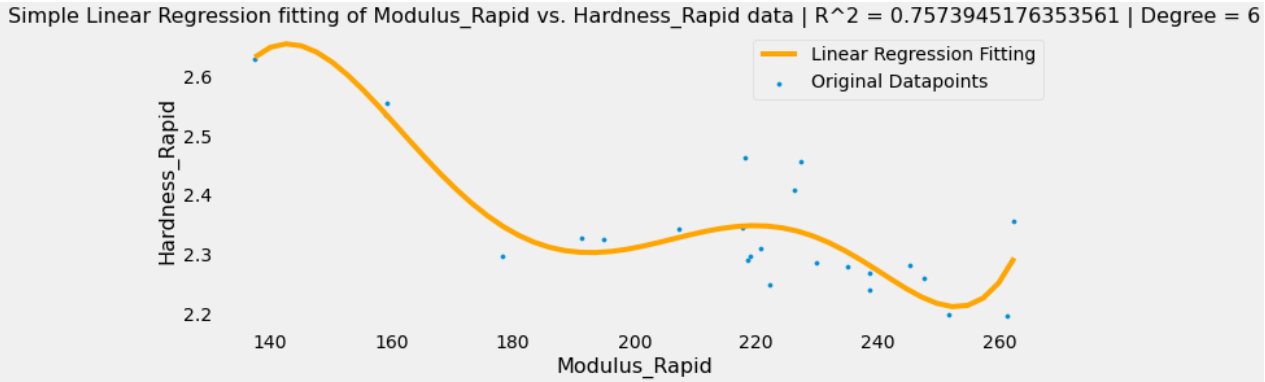
CSM Test

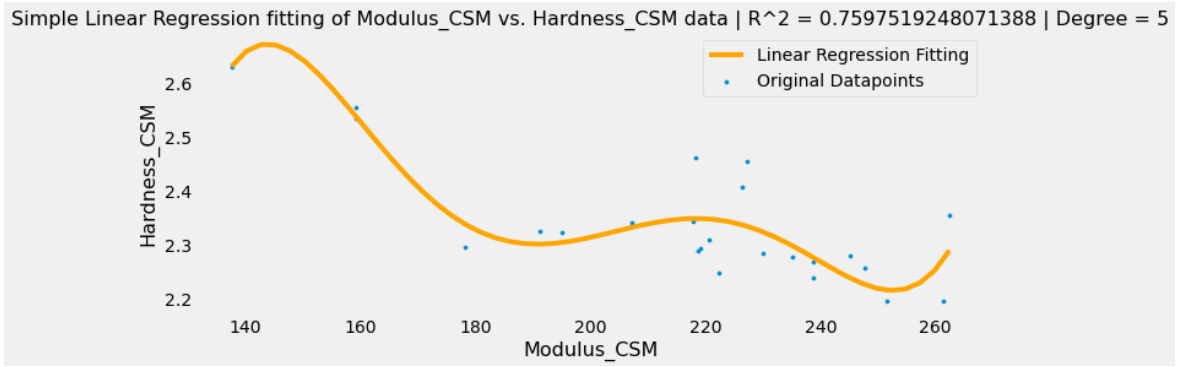




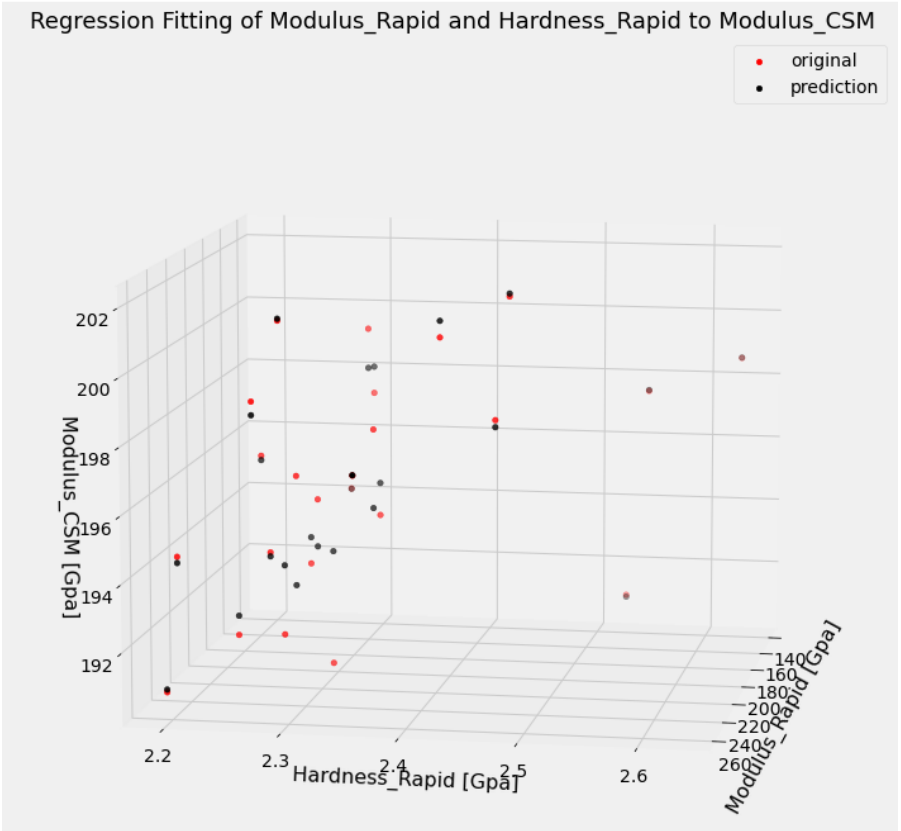
R^2 SCORE = 0.12161215077897891
beta_0, beta_i: 0.28161959905878087 [0.01013807]
Mean Squared Error = 0.008047394954601083

Multilinear and Polynomial regression – hardness vs. modulus (Rapid and CSM)

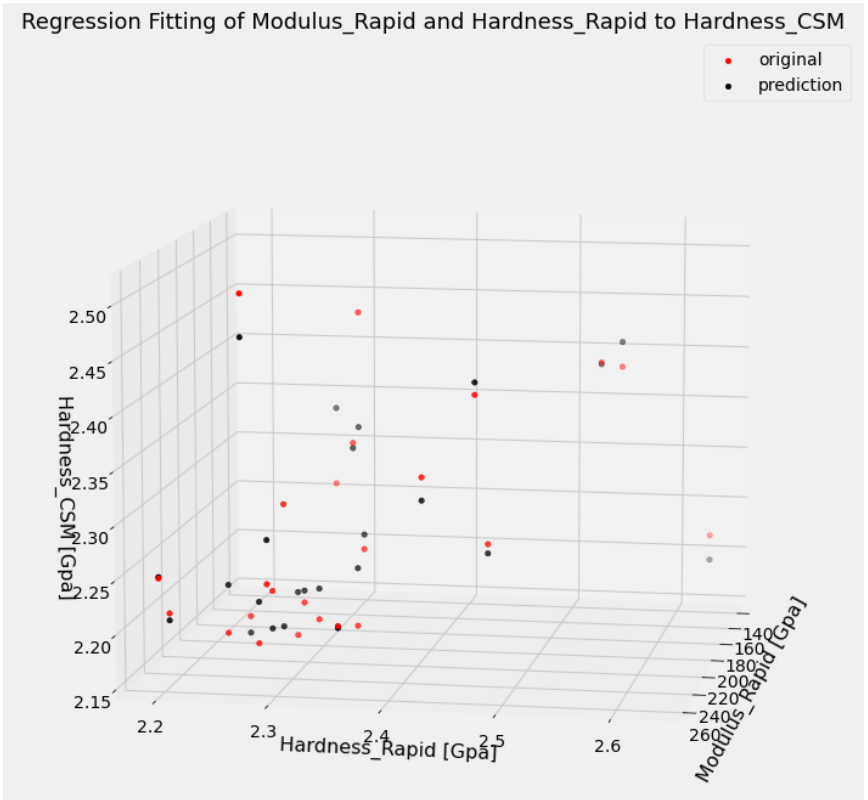




R^2 = 0.851819359315411
MSE = 1.606301194704029



R^2 = 0.7863352580700034
MSE = 0.0019575003999753055



Prediction of Berkovich Hardness and Modulus

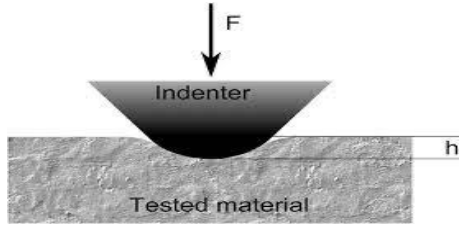
Given Properties:

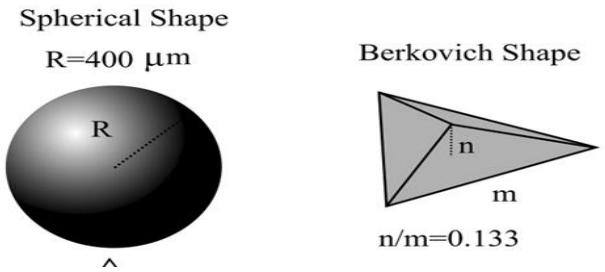
Berkovich Nanohardness and Modulus (only in shock loaded condition)

Spherical Nano Indentation stress at 5% strain, and Spherical modulus (in both asbuilt and shock-loaded condition)

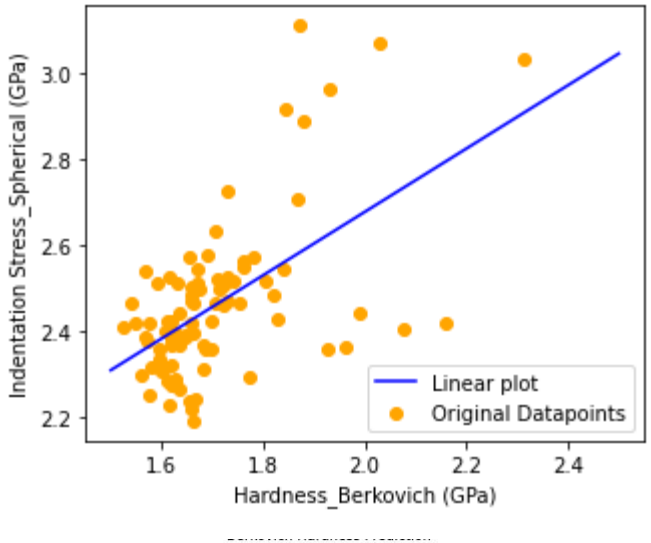
Missing?

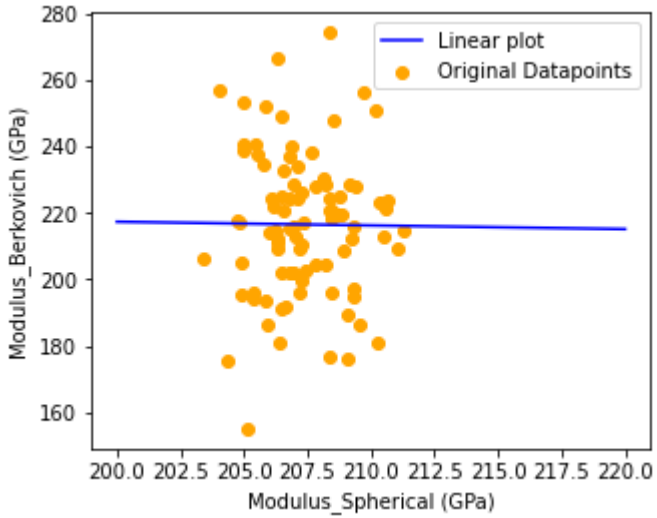
Berkovich Nanohardness and Modulus in as-built condition)

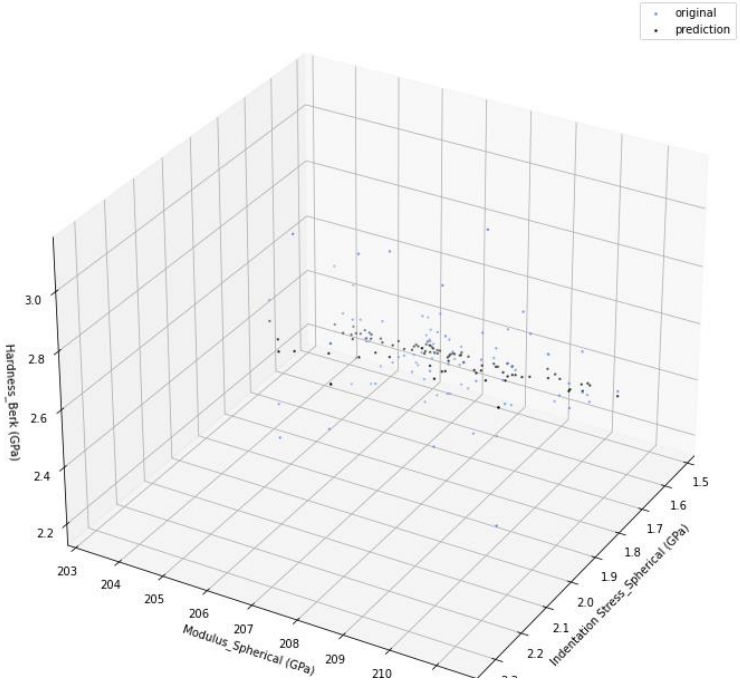


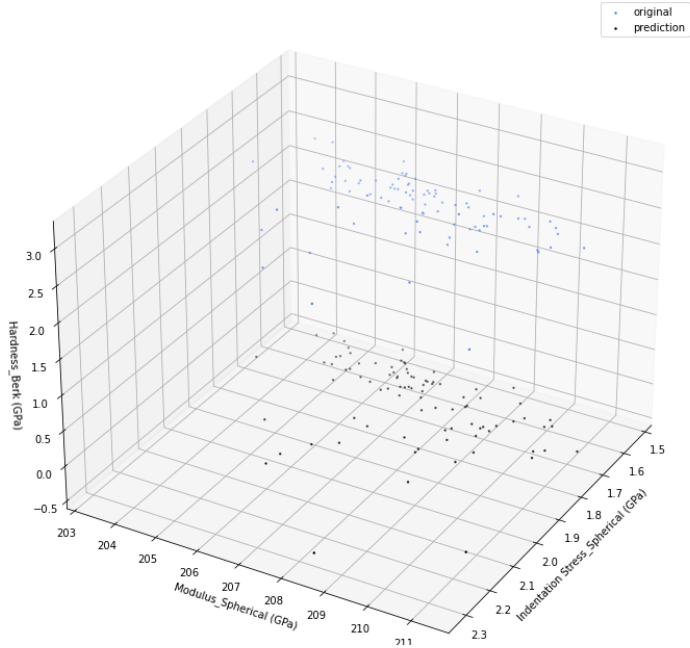


Prediction of Berkovich Hardness and Modulus









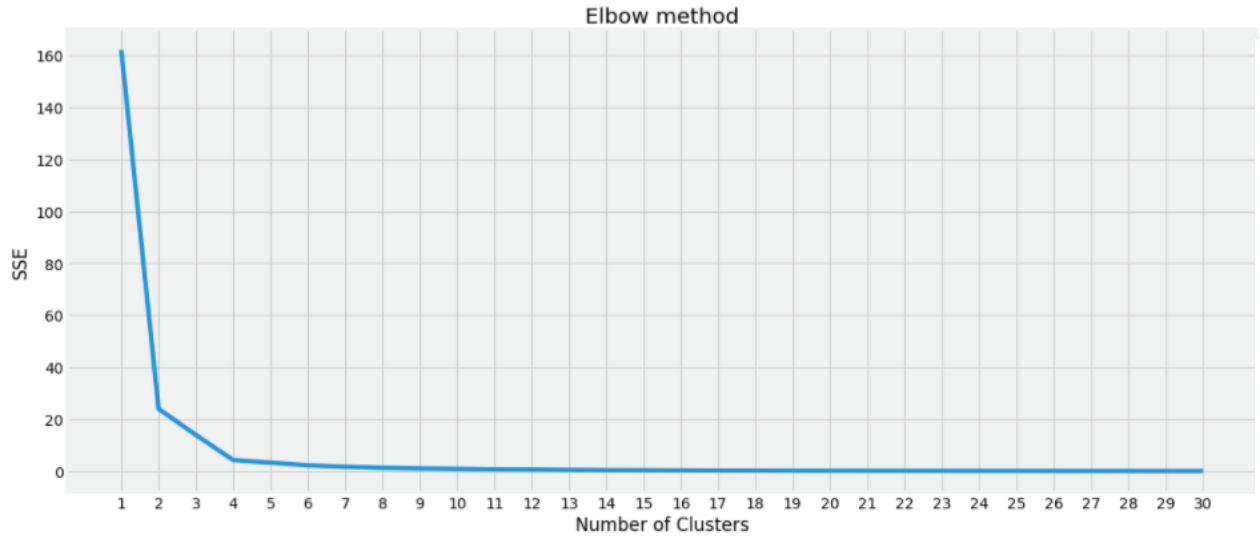
Prediction of Berkovich Hardness and Modulus

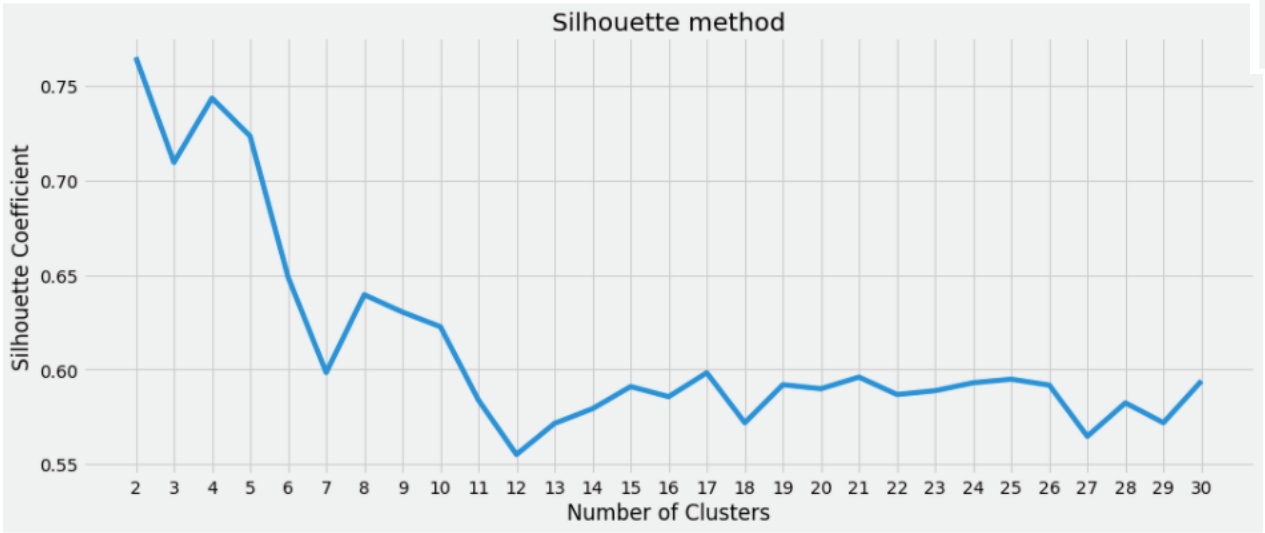
Model	Target	MAE	MSE	RMSE	R^2
Linear model 1	Berkovich Hardness	0.1075	0.02209	0.8187	0.31898
Linear model 2	Berkovich Modulus	16.75	466.47	0.818	7.95×10^{-5}
Multilinear Model 1	Berkovich Hardness	16.76	466.11	0.8188	0.31898
Multilinear Model 2	Berkovich Modulus	0.1075	0.02209	0.8187	8.4×10^{-4}
ANN	Berkovich Modulus and HArdness	In Progress	In Progress	In Progress	In Progress

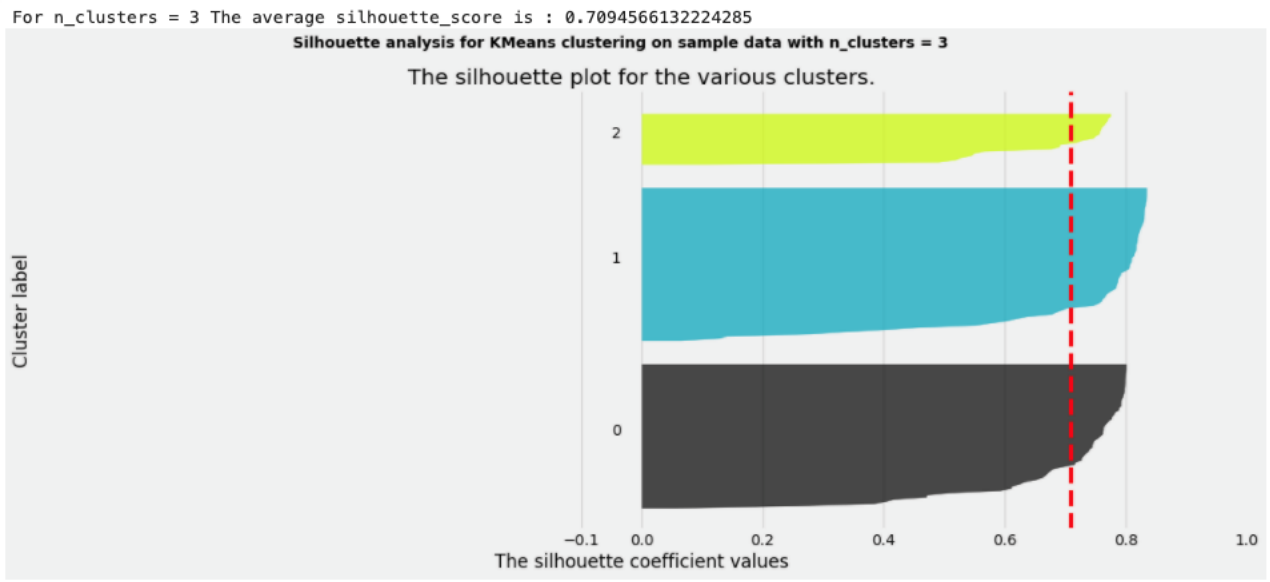
- **Modulus is difficult to model, However Hardness partially works**
- **Average berkovich hardness in as-built condition is 2.13 GPa and Shock loaded condition is 2.46 GPa**
- Shock loading increases the hardness after electron beam printing of Tantalum (plastic deformation due to shock loading)
- Literatures reported tanatlum nano hardness between 2-4.5 GPa, depending on orientation*.

*T.P. Remington, C.J. Ruestes, E.M. Bringa, B.A. Remington, C.H. Lu, B. Kad, M.A. Meyers, Plastic deformation in nanoindentation of tantalum: A new mechanism for prismatic loop formation, Acta Materialia,Volume 78,2014,Pages 378-393,ISSN 1359-6454,https://doi.org/10.1016/j.actamat.2014.06.058.

Unsupervised Machine Learning: Determination of optimal number of clusters with GB1 data based on Hardness value for Asbuilt condition

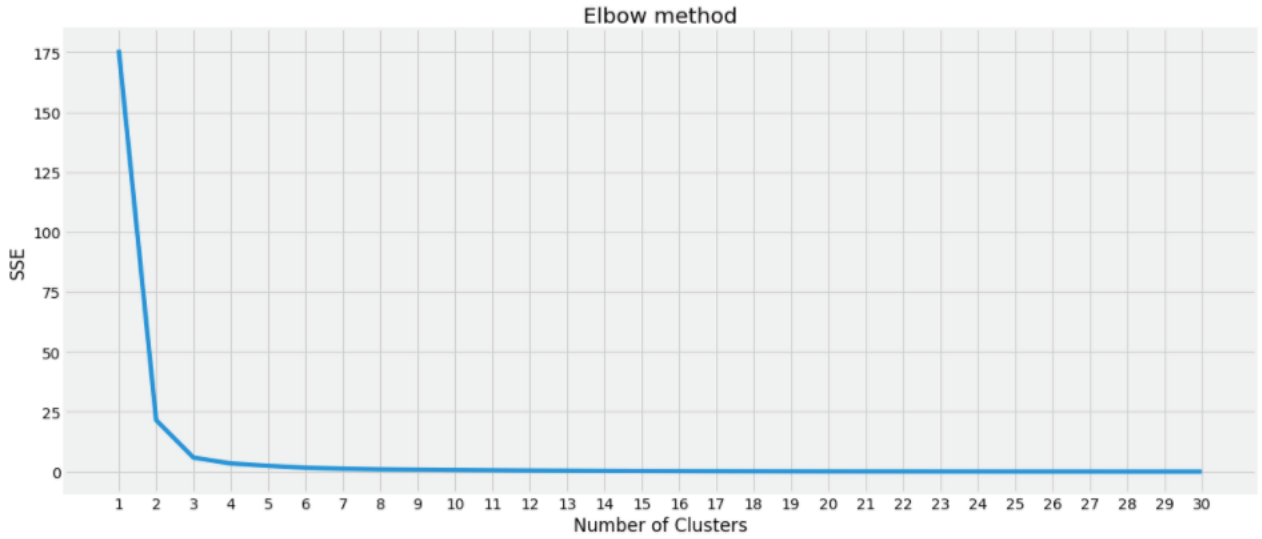


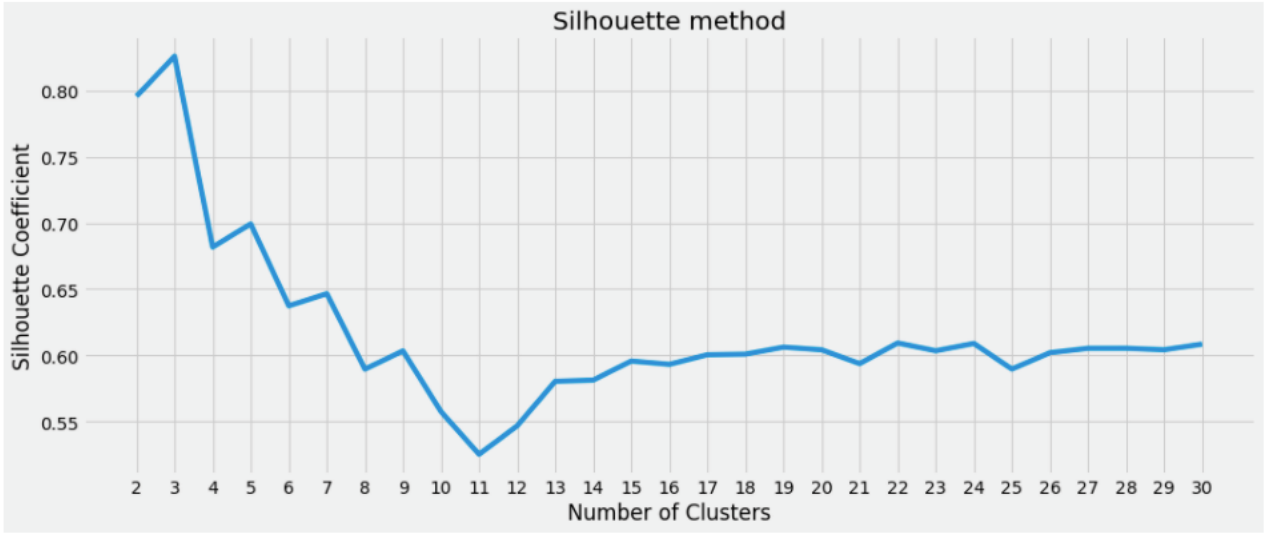


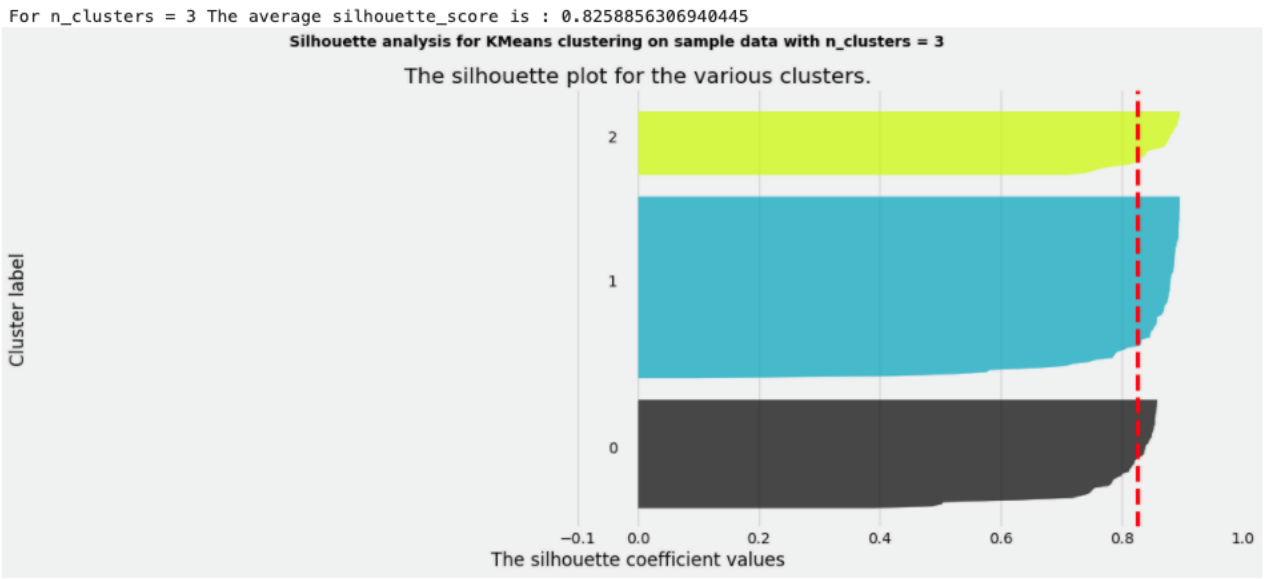


Considered all GB1 data on Asbuilt condition and determined optimal number of clusters using both Elbow method and Silhouette method based on Hardness values.
Optimal number of cluster: 3
This exactly correlates with the experimental results as the data can be classified into:
(1) Berkovich Hardness- Shockloaded,
(2) Spherical Indentation stress- Asbuilt,
(3) Spherical Indentation stress- Shockloaded

Determination of optimal number of clusters with GB2 data based on Hardness value for Asbuilt condition







Considered all GB2 data on Asbuilt condition and determined optimal number of clusters using both Elbow method and Silhouette method based on Hardness values.

Optimal number of cluster: 3

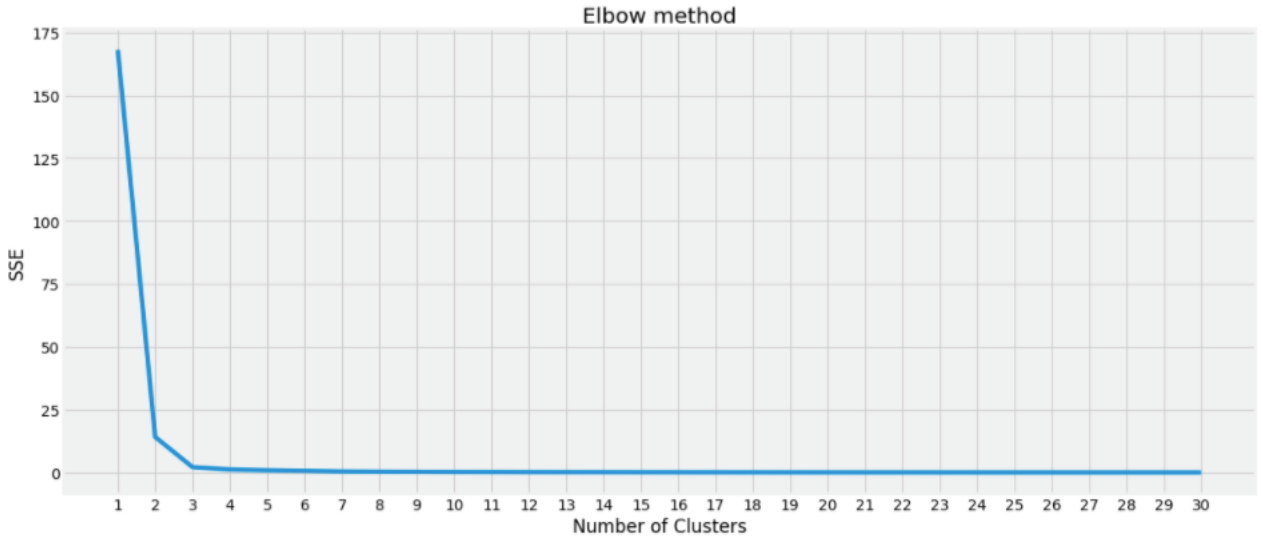
This exactly correlates with the experimental results as the data can be classified into:

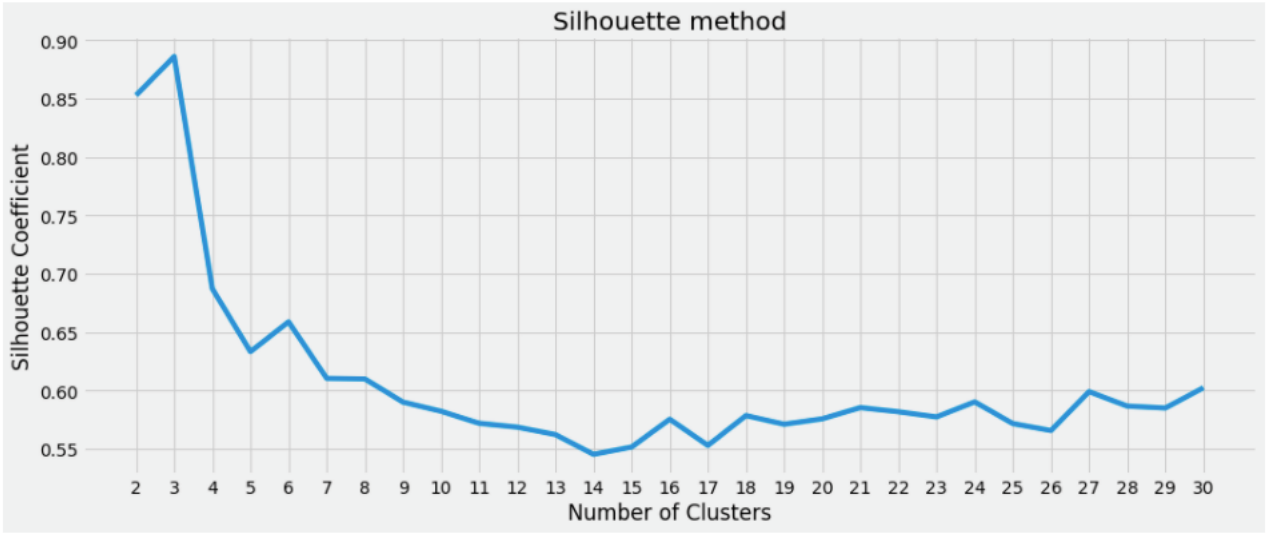
(1) Berkovich Hardness- Shockloaded,

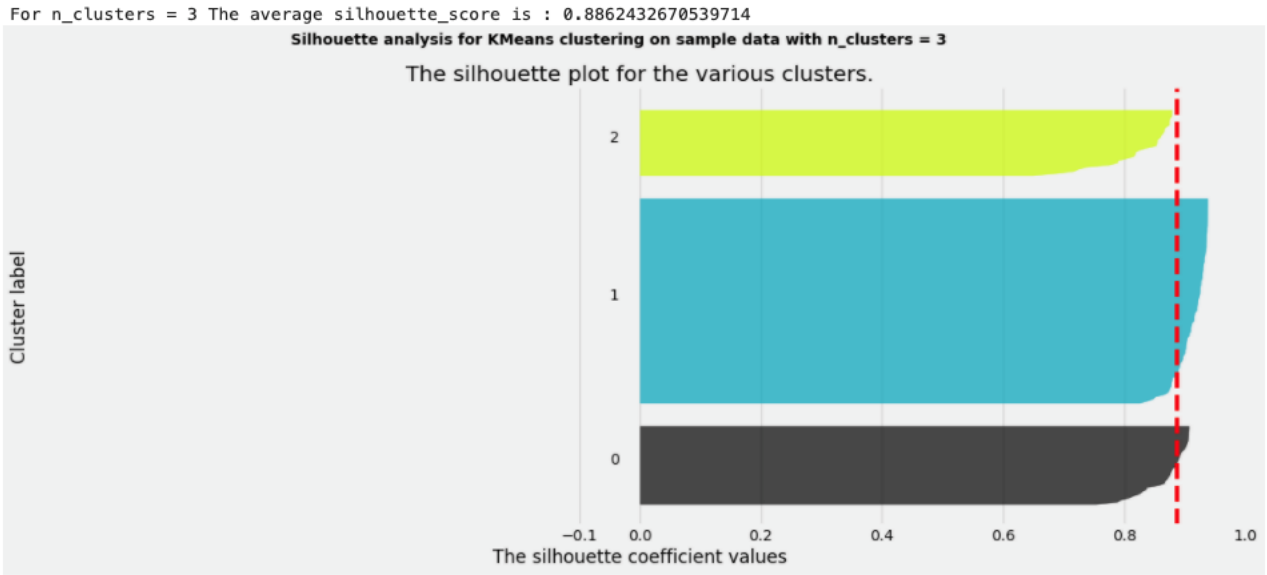
(2) Spherical Indentation stress- Asbuilt,

(3) Spherical Indentation stress- Shockloaded

Determination of optimal number of clusters with GB3 data based on Hardness value for Asbuilt condition







Considered all GB3 data on Asbuilt condition and determined optimal number of clusters using both Elbow method and Silhouette method based on Hardness values.

Optimal number of cluster: 3

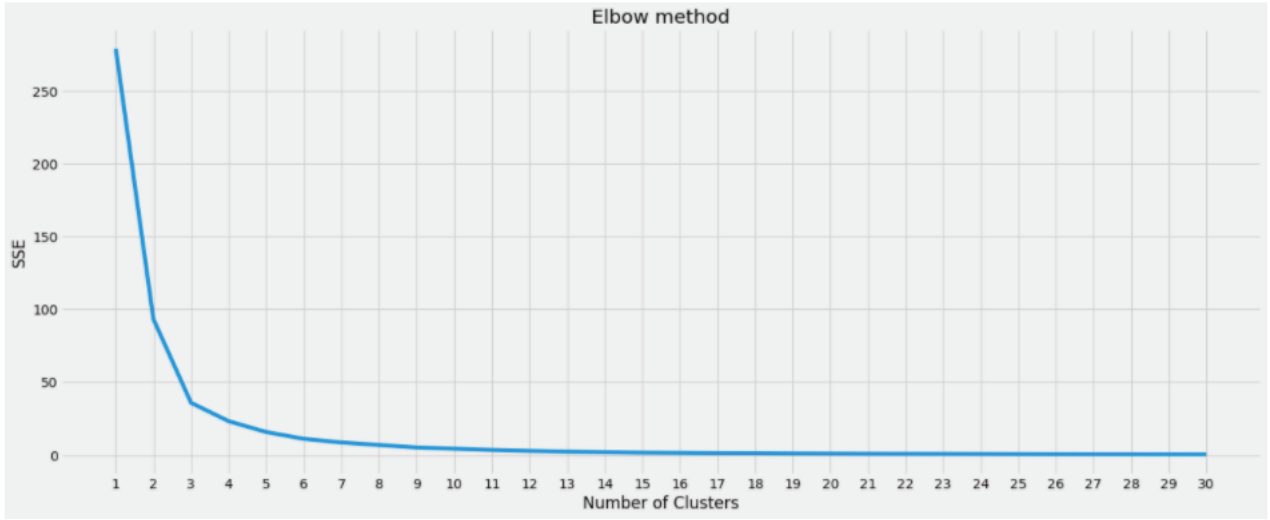
This exactly correlates with the experimental results as the data can be classified into:

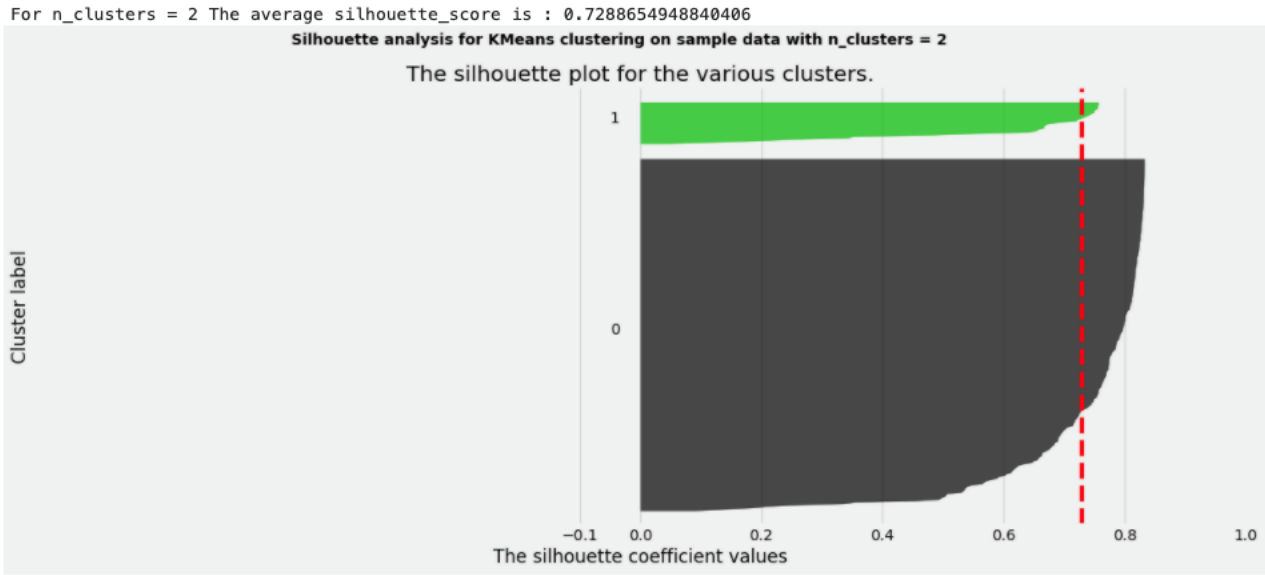
(1) Berkovich Hardness- Shockloaded,

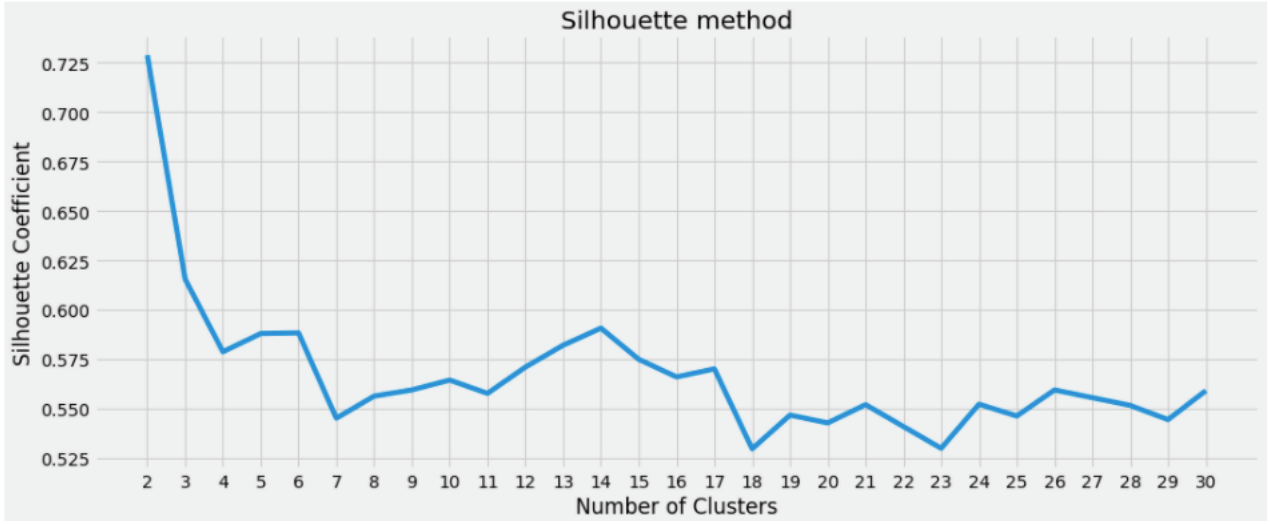
(2) Spherical Indentation stress- Asbuilt,

(3) Spherical Indentation stress- Shockloaded

Determination of optimal number of clusters with GB data based on Hardness value for Berkovich condition







Considered all GB data (GB1, GB2 and GB3) on Berkovich condition and determined optimal number of clusters using both Elbow method and Silhouette method based on Hardness values.

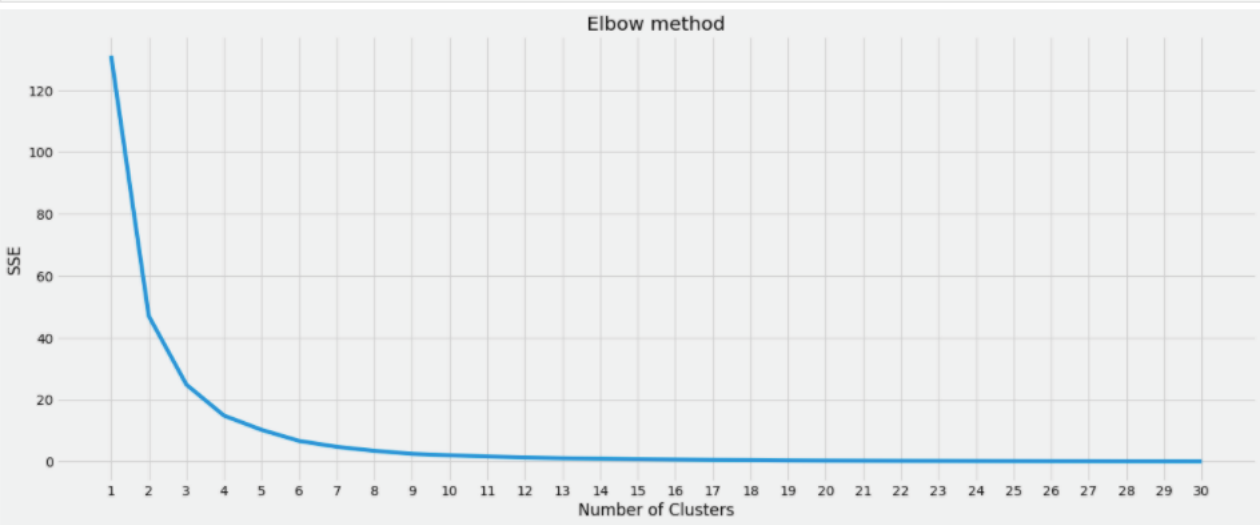
Optimal number of cluster: 2

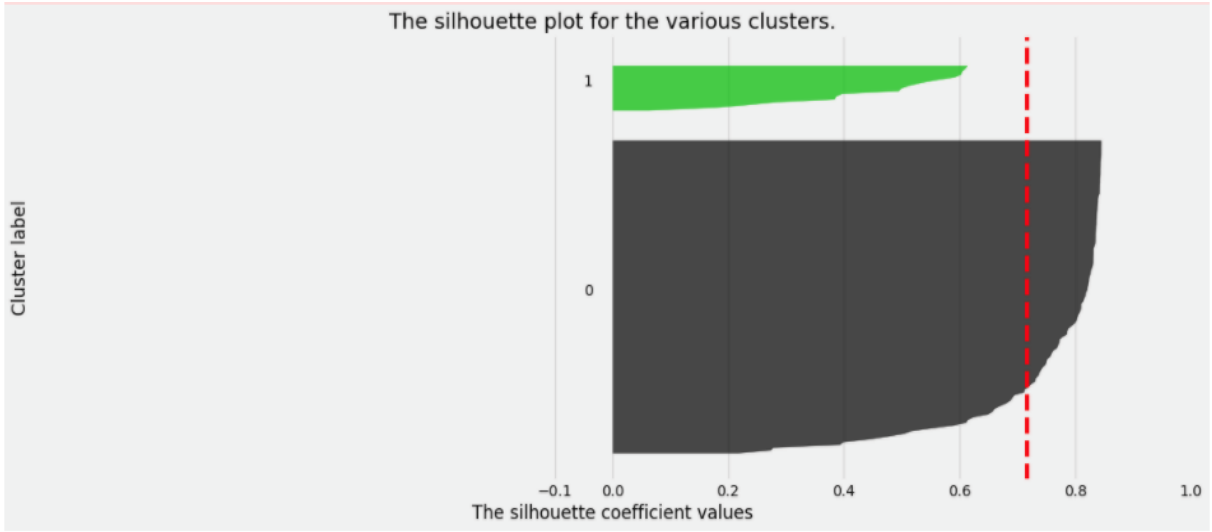
This exactly correlates with the experimental results as the data can be classified into:

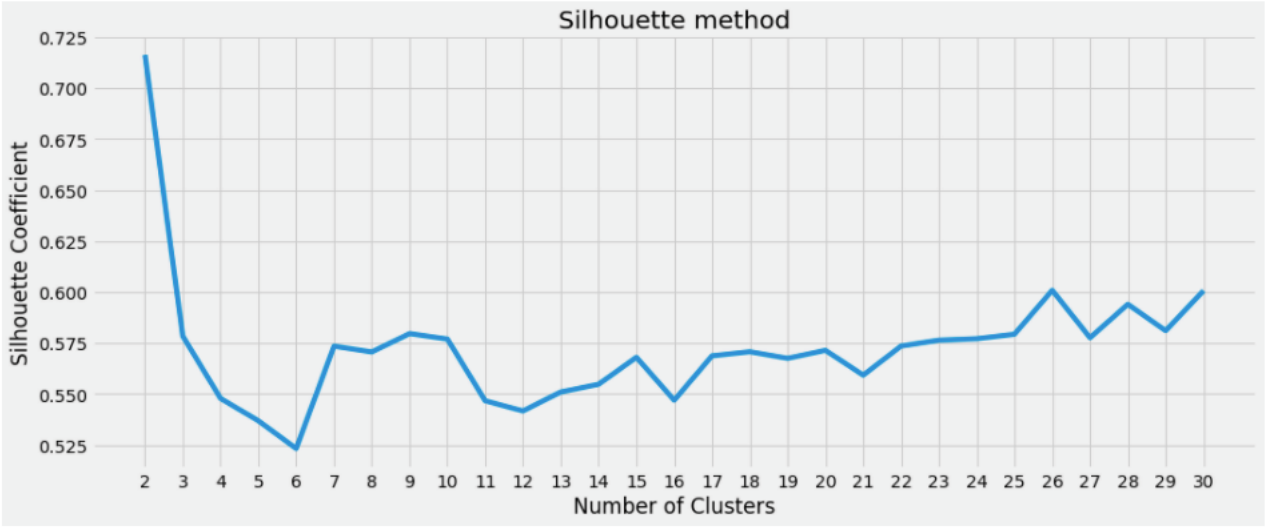
(1) GB1, GB2 - Berkovich Hardness - Shockloaded

(2) GB3 - Berkovich Hardness - Shockloaded

Determination of optimal number of clusters with GB data based on Hardness value for Spherical condition

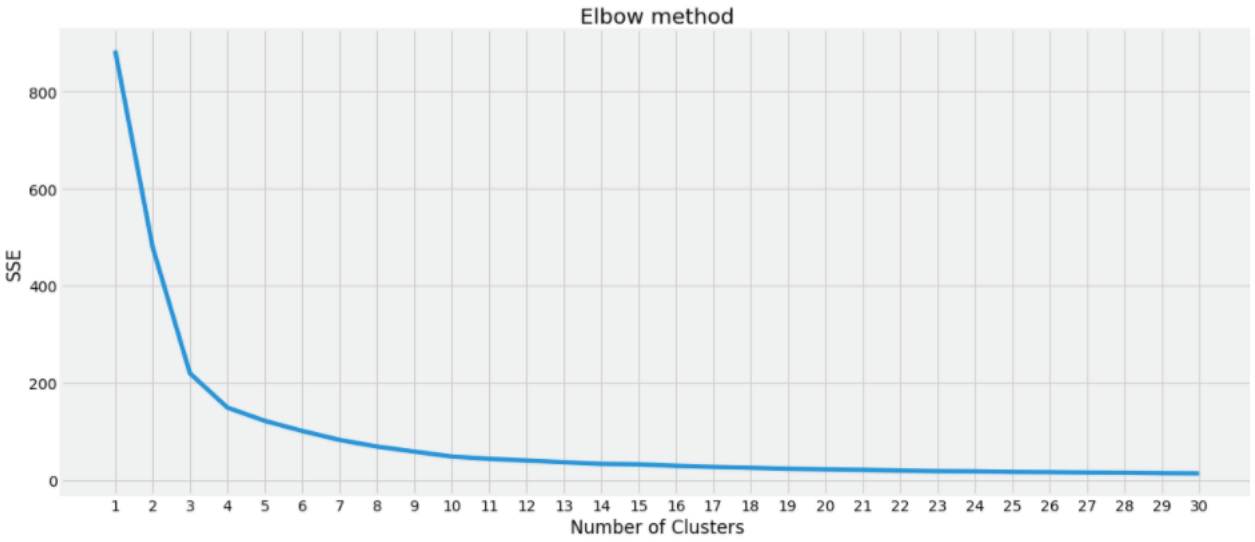


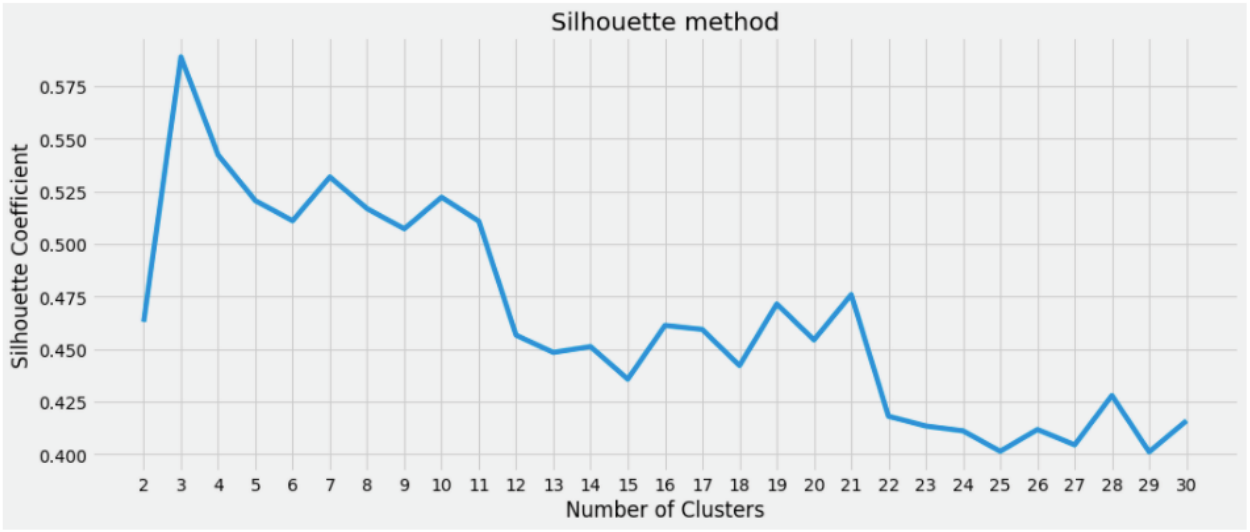


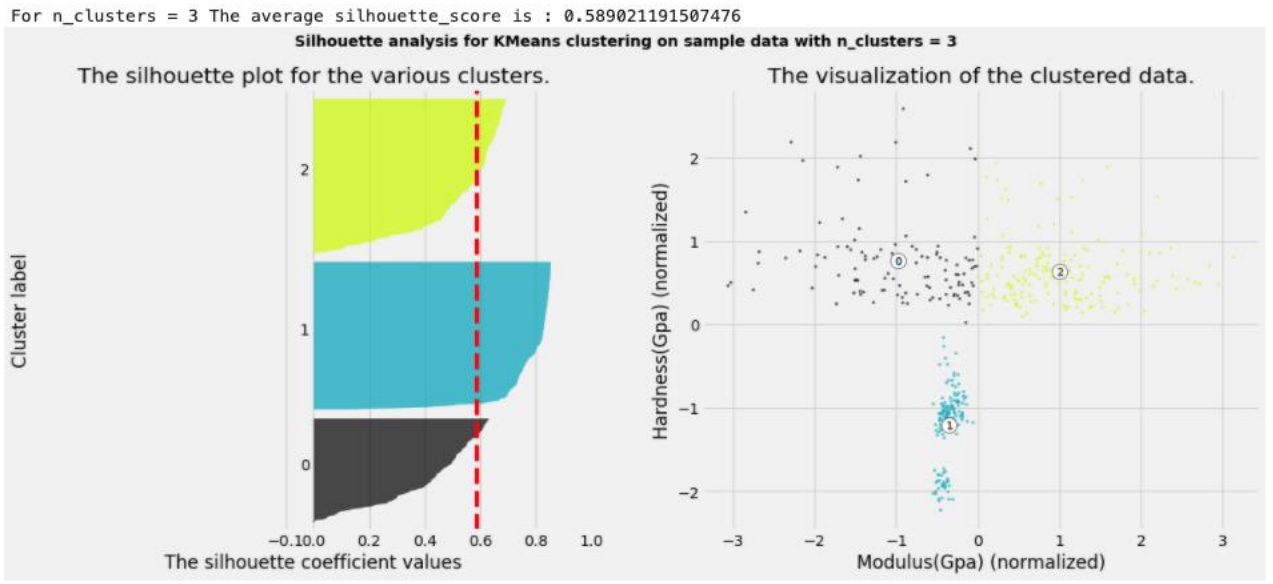


Considered all GB data (GB1, GB2 and GB3)
on Spherical condition and determined optimal number of clusters using both Elbow method and Silhouette method based on Hardness values.
Optimal number of cluster: 2
This exactly correlates with the experimental results
as the data can be classified into:
(1) GB1, GB2 - Spherical Indentation stress - Shockloaded
(2) GB3 - Spherical Indentation stress - Shockloaded

Unsupervised Learning



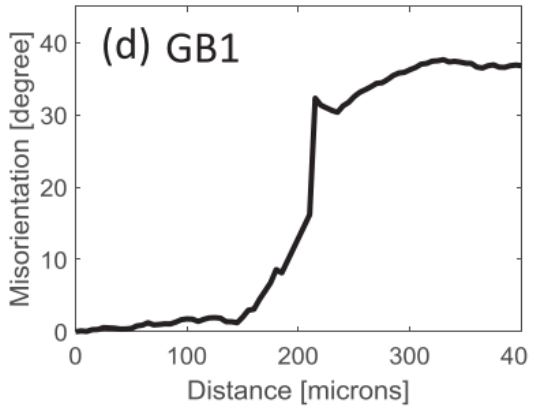


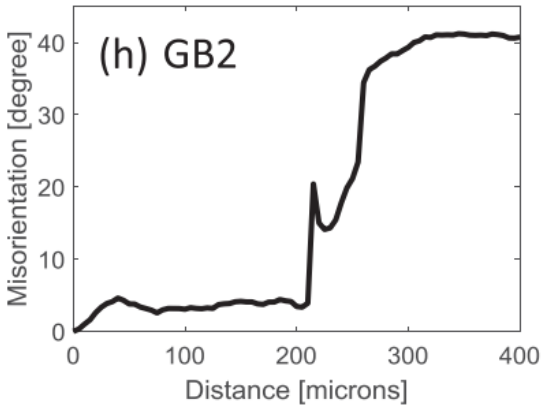


Considered all GB data (GB1, GB2 and GB3) on both spherical and berkovich condition and determined optimal number of clusters using both Elbow method and Silhouette method based on Hardness and modulus values.
Optimal number of cluster: 3
This exactly correlates with the experimental results as the data can be classified into:
(1) Berkovich Hardness- Shockloaded,
(2) Spherical Indentation stress- Asbuilt,
(3) Spherical Indentation stress- Shockloaded

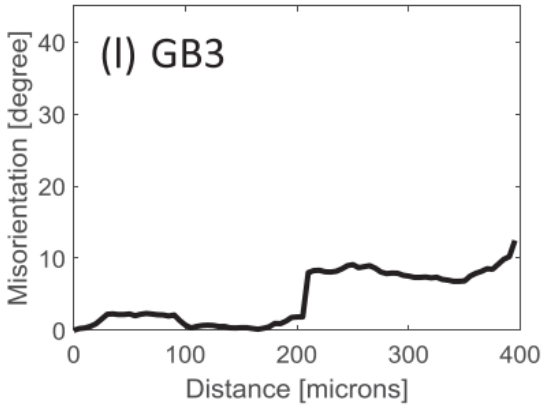
Unsupervised Learning

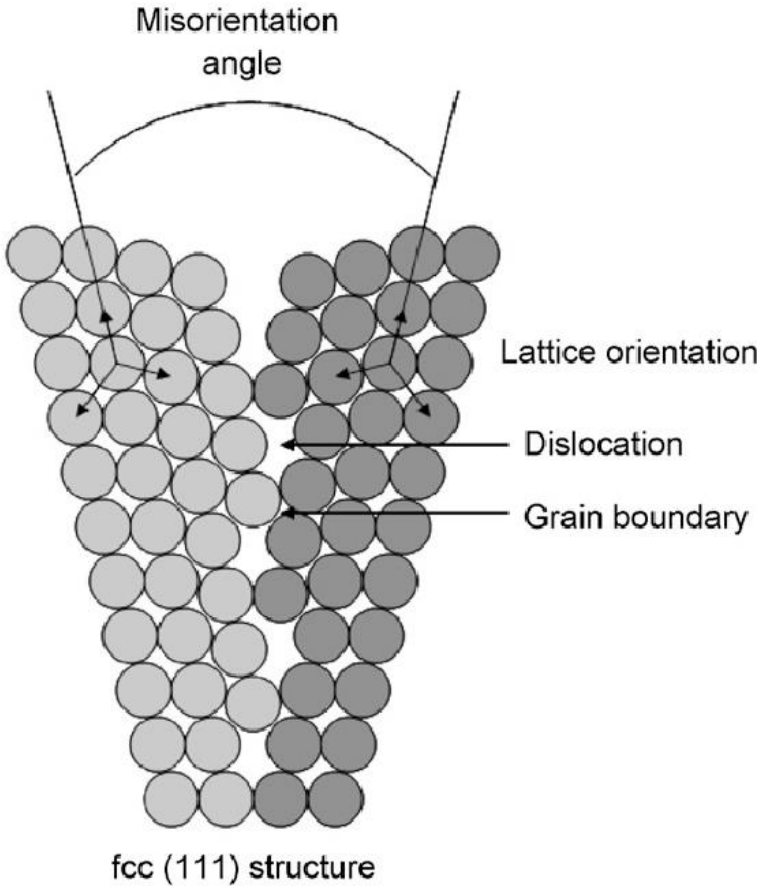
Higher misorientation





Lower misorientation





Conclusion, Dataset and Challenges

Conclusion

- Rapid test and CSM test correlate each other
- Prediction of as-built nano hardness is 2.16 GPa, which is less than shock loaded tantalum.
- Shock loading induces plastic deformation
- The modulus is difficult to model
- The nano-hardness is dependent on misorientation of grains and can be clustered accordingly

Challenges

The data set is limited due to complexity of experiments

The data lengths are not equal for all provided nano indenation

Thank you!

Material Science and Engineering Data Science Project

Mohammad Sayem Bin Abdullah (ME), Yutaro Sugimoto (MSE), Abhay Prithvi Komanduri (ME), Moses Prasad Varghese (ME)

Section - 1: Introduction & Dataset

The dataset was acquired through the Mendeley database which was stored as supplemental information for the paper "Quantifying heterogeneous deformation in grain boundary regions on shock loaded Tantalum using spherical and sharp tip nanoindentation." [1] The dataset was generated by a series of nanoindentation tests performed on tantalum (Ta) samples created using the electron beam wire feed technique, which is one of the additive manufacturing (AM) methods. Two kinds of samples were prepared, as-built and shock-loaded, according to the treatment post-production. Researchers performed nanoindentation tests across three grain boundary regions - GB1, GB2 and GB3. The grain boundary locations were selected based on their position from spalling damage and void. GB1 and GB2 are high angle grain boundaries, while GB3 is lower angle grain boundary. In the lower angle grain boundary, the two adjacent crystals are oriented at less than 5 ❑°, while it is greater than 5 ❑° for the higher angle grain boundary. The voids were preferentially distributed along the grain boundary in GB1 and GB2, whereas the voids were distributed away from the grain boundary in GB3. In addition to the type of samples and GB angles, datasets are also classified by the type of tips used for nanoindentation tests, which were Berkovich and spherical tips, as well as by the speed of indentation (Rapid vs. CSM (continuous stiffness measurement)). Multiple tests (~60) were performed across each grain boundary within $100\,\mu m$ region where hardness and distance (tip travel distance) were recorded for each test.

We hypothesized that hardness values from the rapid indentation test and CSM indentation test have a correlation. To verify this hypothesis, we apply the linear regression method and calculated the error (parity plots and residual plots).

We also hypothesized that hardness value and nanoindentation stress value have a correlation. We apply machine learning methods such as (regression, SVM, KNN) to verify this idea. Qiao et al. (2021) [2] used the KNN model for this purpose in their study on glass materials, but we hope to apply other methods for the clearer outcome.

Greater defect density is expected in a shock-loaded sample since impact and resulting deformation introduces defects in the material. This is faintly observed in the stress-strain graph, but by applying machine learning methods, we hope to reveal the truth more clearly.

For a more challenging task, we try to find the relationship between nanoindentation behavior and grain boundary types and spalling damage. In the hardness vs. modulus plot, data clustering will be observed. Koumoulos et al (2019) [3] showed similar plot on their research on composite.

Section - 2: Method

First, for both rapid and CSM indentation tests, we applied the linear regression method to see the correlation between hardness and modulus value, and calculated the error. Regression fitting was performed on load and displacement to modulus and to hardness. In addition to this,

we have also performed polynomial regression for the modulus_rapid and hardness_rapid to modulus_CSM and hardness_CSM. We have also optimized the multi-polynomial regression for comparing the various parameters, and found that the prediction of Hardness_CSM from Hardness_Rapid and Modulus_Rapid is much better than the prediction of Modulus_CSM from the same two features.

Section - 3: Results & Conclusion

Results have been discussed after each methodology.

References

1. Jordan S. Weaver, David R. Jones, Nan Li, Nathan Mara, Saryu Fensin, George T. Gray, Quantifying heterogeneous deformation in grain boundary regions on shock loaded tantalum using spherical and sharp tip nanoindentation, Materials Science and Engineering: A, Volume 737, 2018, Pages 373-382, ISSN 0921-5093, https://doi.org/10.1016/j.msea.2018.09.075.
2. Qian Qiao, Hongtu He, Jiaxin Yu, Yafeng Zhang, Huimin Qi, Applicability of machine learning on predicting the mechanochemical wear of the borosilicate and phosphate glass, Wear, Volume 476, 2021, 203721, ISSN 0043-1648, https://doi.org/10.1016/j.wear.2021.203721
3. Koumoulos, E.; Konstantopoulos, G.; Charitidis, C. Applying Machine Learning to Nanoindentation Data of (Nano-) Enhanced Composites. Fibers 2020, 8, 3. https://doi.org/10.3390/fib8010003

Note: Please go through the code sequentially. We have all developed the code independently and few variables might be same, and hence if you skip the blocks, you might not get the accurate results

```
# Installing modules for the analysis
!pip install kneed
!pip install minisom
```

```
Requirement already satisfied: kneed in /opt/conda/lib/python3.8/site-
packages (0.7.0)
Requirement already satisfied: scipy in /opt/conda/lib/python3.8/site-
packages (from kneed) (1.6.3)
Requirement already satisfied: matplotlib in
/opt/conda/lib/python3.8/site-packages (from kneed) (3.4.1)
Requirement already satisfied: numpy>=1.14.2 in
/opt/conda/lib/python3.8/site-packages (from kneed) (1.20.2)
Requirement already satisfied: cycler>=0.10 in
/opt/conda/lib/python3.8/site-packages (from matplotlib->kneed)
(0.10.0)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.8/site-packages (from matplotlib->kneed)
(2.8.1)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/conda/lib/python3.8/site-packages (from matplotlib->kneed)
(2.4.7)
```

```
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.8/site-packages (from matplotlib->kneed)
(1.3.1)
Requirement already satisfied: pillow>=6.2.0 in
/opt/conda/lib/python3.8/site-packages (from matplotlib->kneed)
(8.1.2)
Requirement already satisfied: six in /opt/conda/lib/python3.8/site-
packages (from cycler>=0.10->matplotlib->kneed) (1.15.0)
Requirement already satisfied: minisom in
/opt/conda/lib/python3.8/site-packages (2.2.9)
```

```
# Importing all the important modules for the analysis
import numpy as np
import seaborn as sns
import pandas as pd
import pickle

import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from matplotlib.patches import Rectangle
from matplotlib.patches import Circle
from matplotlib.patches import Arrow
import matplotlib.cm as cm
from mpl_toolkits import mplot3d
from mpl_toolkits.mplot3d import Axes3D

import time
import scipy as sp
from scipy import stats

import sklearn
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split, cross_val_score,
KFold
from sklearn.preprocessing import StandardScaler, LabelEncoder,
LabelBinarizer
from sklearn.model_selection import RandomizedSearchCV
from sklearn.cluster import KMeans
from sklearn.metrics import mean_squared_error, r2_score,
silhouette_samples, silhouette_score, mean_absolute_error
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier

from kneed import KneeLocator

import minisom
from minisom import MiniSom
```

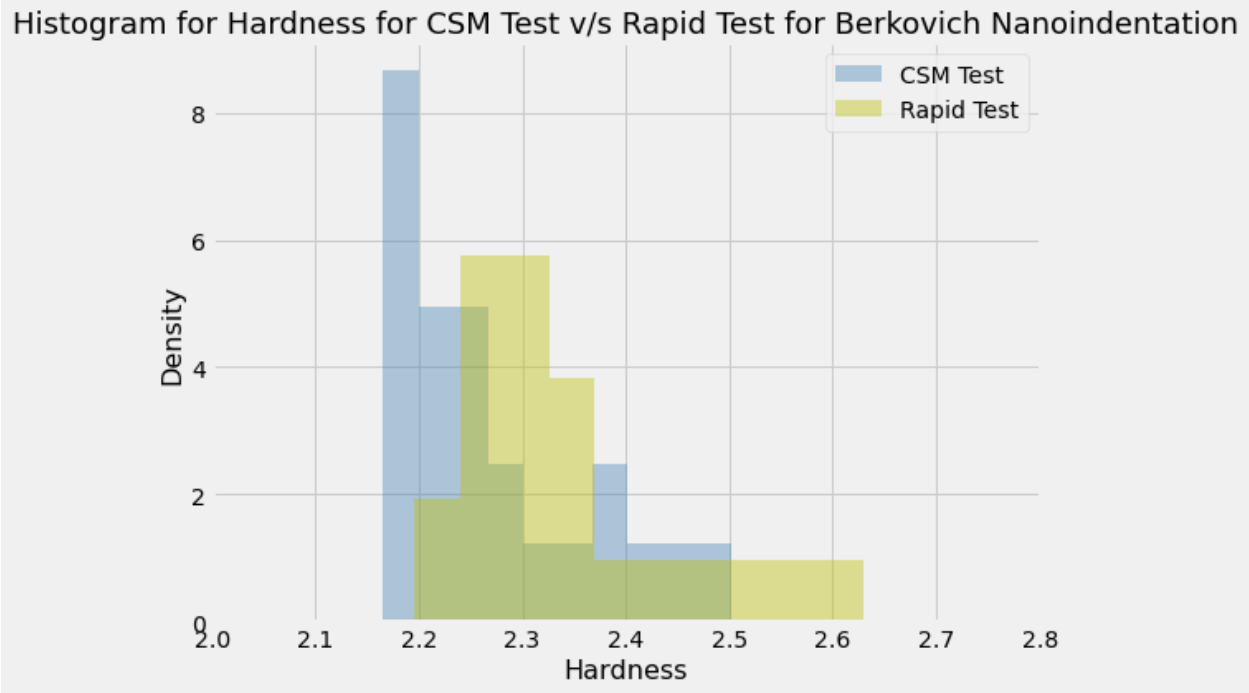
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import Input
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

import random as python_random

plt.style.use('fivethirtyeight')
%matplotlib inline

COLOR = 'black'
plt.rcParams['text.color'] = COLOR
plt.rcParams['axes.labelcolor'] = COLOR
plt.rcParams['xtick.color'] = COLOR
plt.rcParams['ytick.color'] = COLOR

# This dataset contains the Displacement, Load, Modulus_Rapid,
Hardness_Rapid, Modulus_CSM, Hardness_CSM which is used for the linear
regression,
# multilinear regression, polynomial regression and multi-polynomial
regression to find the correlation between the hardness and modulus
results
# from Rapid and CSM testing
CSM_Rapid_Test = pd.read_csv('CSM_Rapid_Test.csv')
CSM_Rapid_Test
```

	Test No	Displacement [nm]	Load [mN]	Modulus_Rapid [Gpa]	\
0	1	226.331269	3.018523	218.648078	
1	2	225.691328	2.955543	222.298681	
2	3	226.032499	2.924516	261.217411	
3	4	227.658889	3.006769	238.615863	
4	5	226.177057	3.033316	191.263455	
5	6	223.792691	2.981520	220.654192	
6	7	227.653345	3.041532	238.631125	
7	8	227.406900	3.053779	219.056120	
8	9	227.324567	3.042597	235.013848	
9	10	227.696868	3.108294	207.234807	
10	11	223.464882	3.093661	226.331729	
11	12	221.265760	3.099453	159.145388	
12	13	221.101403	2.997298	262.380657	
13	14	227.305742	3.010736	178.251115	
14	15	222.209036	2.934262	194.995129	
15	16	223.708859	3.158371	227.247220	
16	17	219.442675	3.072685	159.248798	
17	18	223.767828	2.957285	229.898478	
18	19	224.545970	2.958187	247.601946	
19	20	225.720993	3.012228	245.199287	

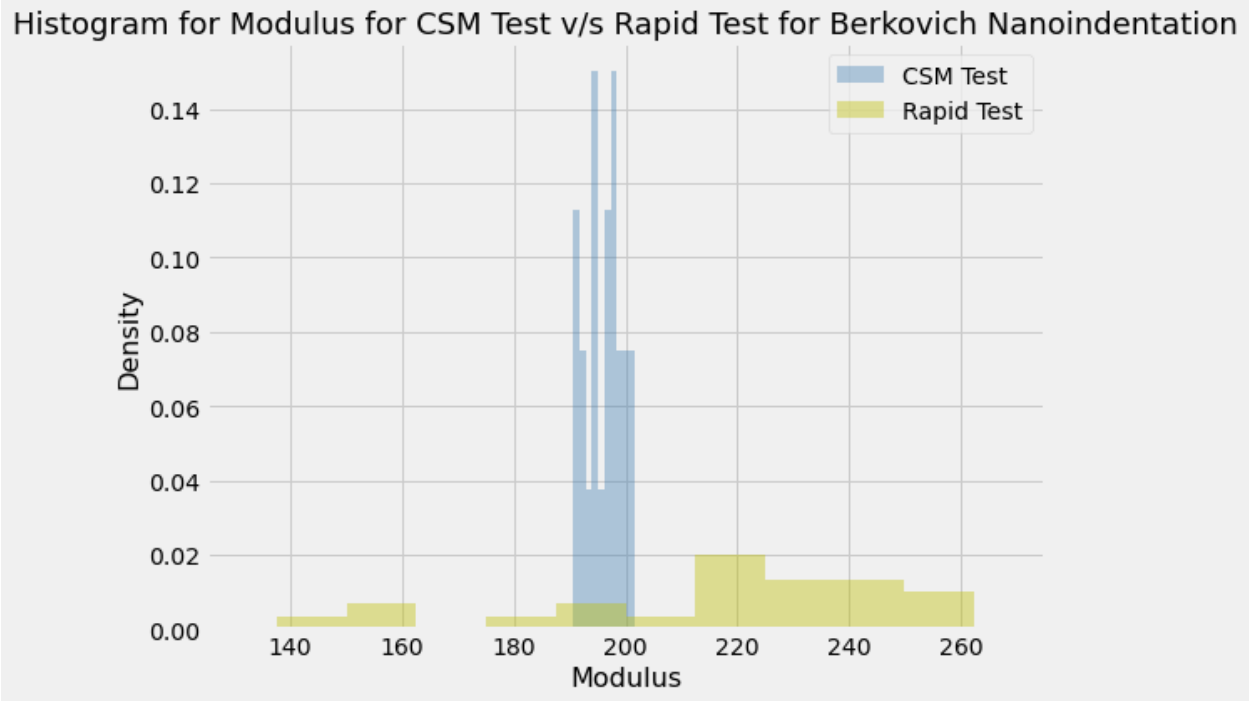
```
20       21         226.653864   3.092601           217.806820
21       22         221.247930   3.154662           137.591688
22       23         221.452521   3.098279           218.155979
23       24         878.894050  40.244531           251.602578
```

	Hardness_Rapid [Gpa]	Modulus_CSM [Gpa]	Hardness_CSM [Gpa]
0	2.289626	193.592881	2.165800
1	2.248734	196.873413	2.185068
2	2.196132	190.669981	2.247373
3	2.240842	191.855585	2.181327
4	2.327044	198.247871	2.457889
5	2.310637	190.643076	2.183133
6	2.268745	194.357783	2.172617
7	2.295866	195.542744	2.197426
8	2.278364	191.830395	2.219549
9	2.343023	194.865566	2.241697
10	2.407838	200.624056	2.324688
11	2.535878	191.512434	2.398039
12	2.356733	197.233745	2.210332
13	2.297216	195.005979	2.284124
14	2.324399	200.266188	2.334856
15	2.456214	198.244493	2.403570
16	2.555573	197.923976	2.394481
17	2.284916	196.465027	2.297794
18	2.259598	198.990090	2.502444
19	2.280870	201.328877	2.233381
20	2.344164	197.668662	2.176105
21	2.629948	198.623815	2.216653
22	2.463119	201.738858	2.258263
23	2.197830	194.441908	2.207838

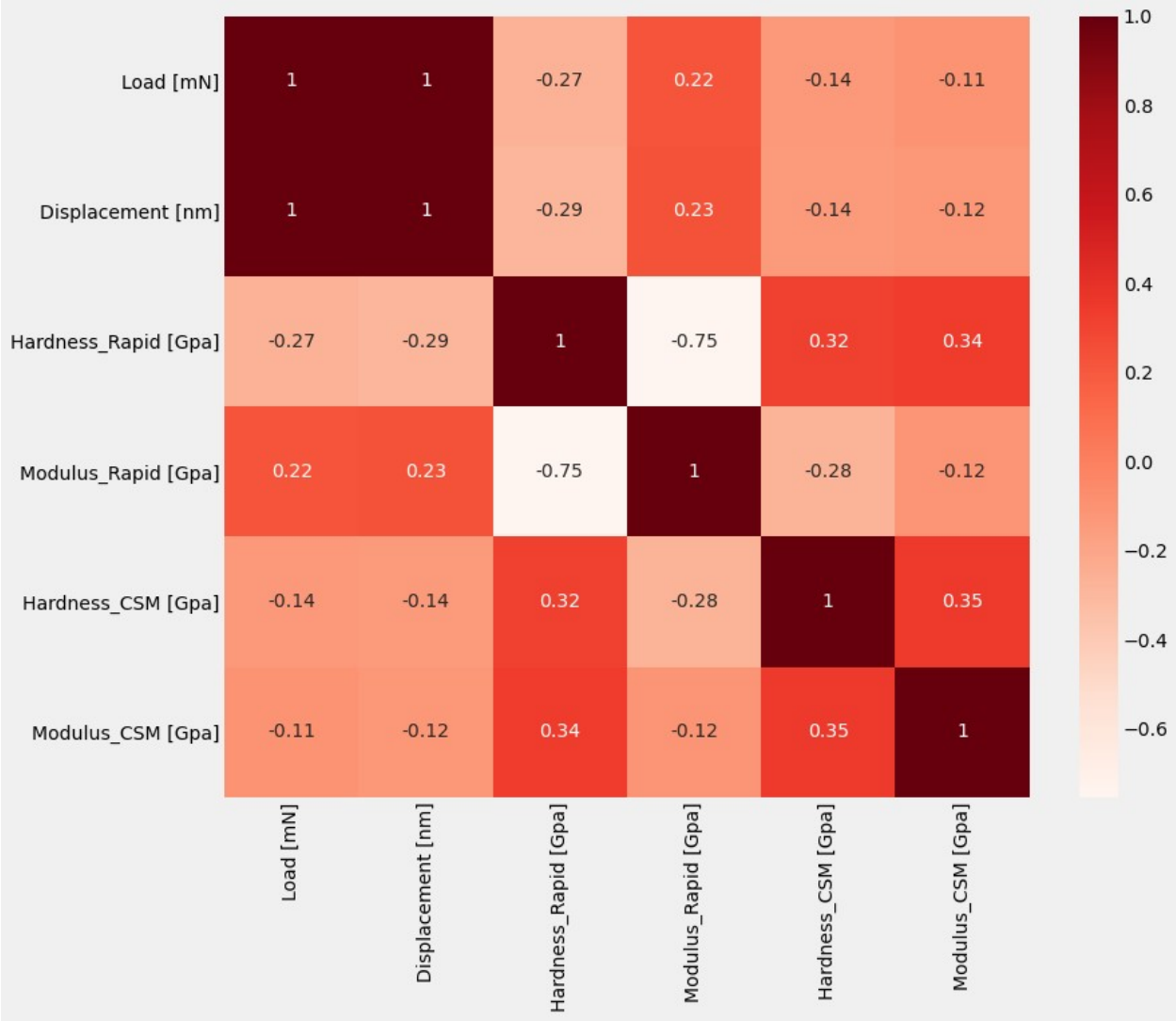
```
plt.figure(figsize=(8,6))
plt.hist(CSM_Rapid_Test['Hardness_CSM [Gpa]'], color = 'steelblue',
bins=10, range=[CSM_Rapid_Test['Hardness_CSM [Gpa]'].min(),
CSM_Rapid_Test['Hardness_CSM [Gpa]'].max()], align='mid', alpha = 0.4,
density=True, label='CSM Test')
plt.hist(CSM_Rapid_Test['Hardness_Rapid [Gpa]'], color = 'y', bins=10,
range=[CSM_Rapid_Test['Hardness_Rapid [Gpa]'].min(),
CSM_Rapid_Test['Hardness_Rapid [Gpa]'].max()], align='mid', alpha =
0.4, density=True, label='Rapid Test')

plt.legend()
plt.xlim([2, 2.8])
plt.ylabel('Density', fontsize=16)
plt.xlabel('Hardness', fontsize=16)
plt.title('Histogram for Hardness for CSM Test v/s Rapid Test for
Berkovich Nanoindentation', fontsize=18)

Text(0.5, 1.0, 'Histogram for Hardness for CSM Test v/s Rapid Test for
Berkovich Nanoindentation')
```




```
plt.figure(figsize=(8,6))
plt.hist(CSM_Rapid_Test['Modulus_CSM [Gpa]'], color = 'steelblue',
bins=10, range=[CSM_Rapid_Test['Modulus_CSM [Gpa]'].min(),
CSM_Rapid_Test['Modulus_CSM [Gpa]'].max()], align='mid', alpha = 0.4,
density=True, label='CSM Test')
plt.hist(CSM_Rapid_Test['Modulus_Rapid [Gpa]'], color = 'y', bins=10,
range=[CSM_Rapid_Test['Modulus_Rapid [Gpa]'].min(),
CSM_Rapid_Test['Modulus_Rapid [Gpa]'].max()], align='mid', alpha =
0.4, density=True, label='Rapid Test')

plt.legend()
plt.xlim([125, 275])
plt.ylabel('Density', fontsize=16)
plt.xlabel('Modulus', fontsize=16)
plt.title('Histogram for Modulus for CSM Test v/s Rapid Test for
Berkovich Nanoindentation', fontsize=18)

Text(0.5, 1.0, 'Histogram for Modulus for CSM Test v/s Rapid Test for
Berkovich Nanoindentation')
```

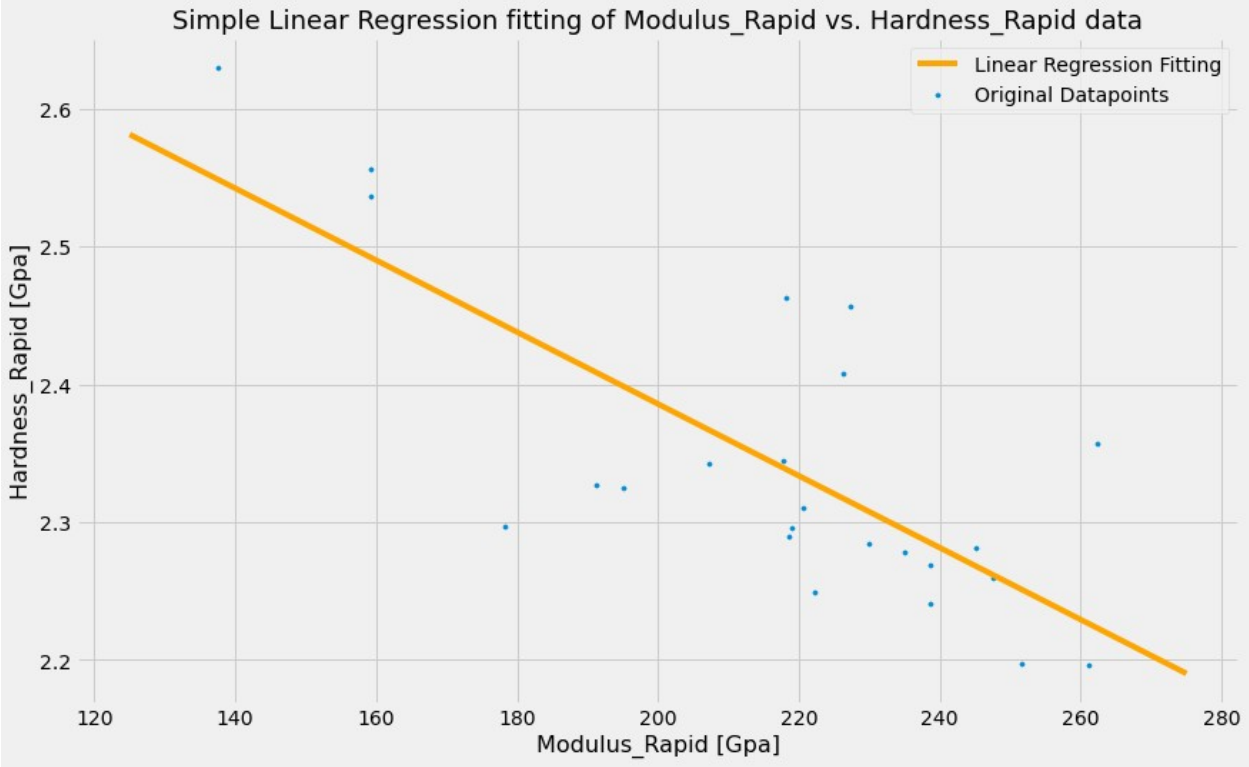


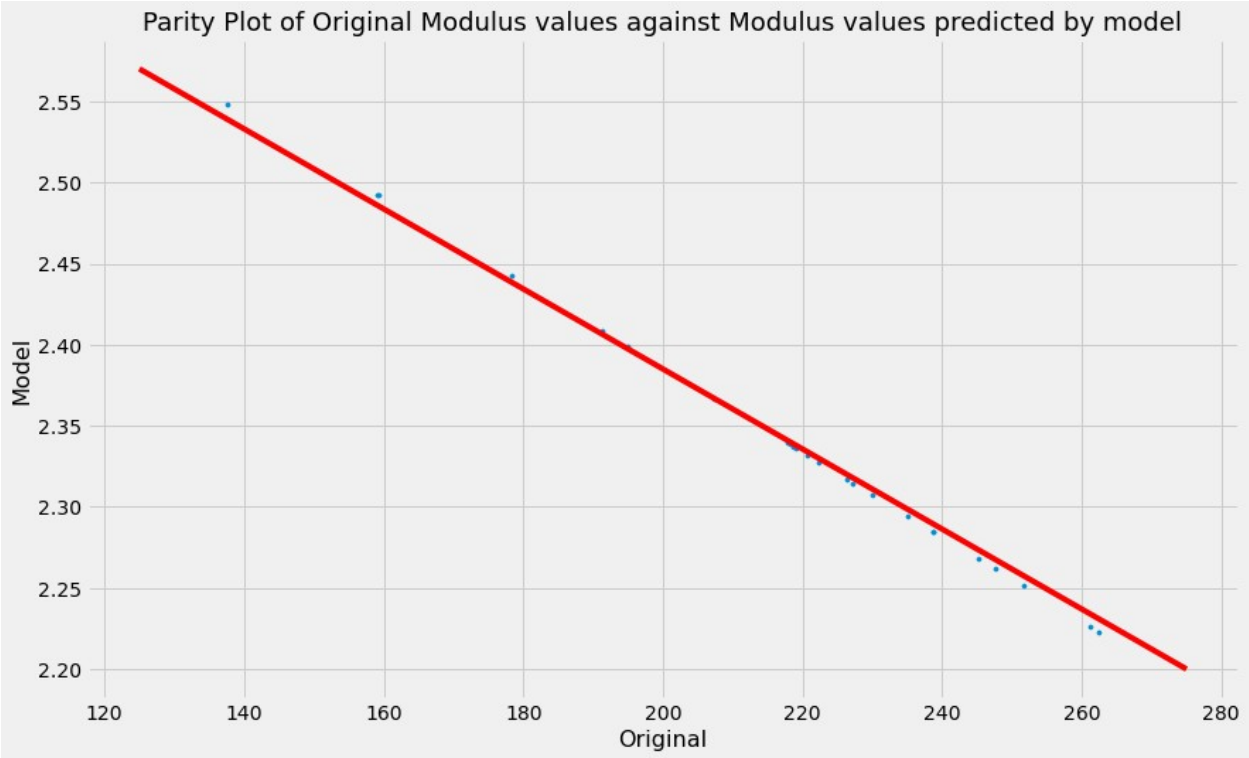

```
cor_Data_CSM_Rapid = CSM_Rapid_Test[['Load [mN]', 'Displacement [nm]',
'Hardness_Rapid [Gpa]', 'Modulus_Rapid [Gpa]', 'Hardness_CSM [Gpa]',
'Modulus_CSM [Gpa]']]

plt.figure(figsize=(12,10))
cor_Data = cor_Data_CSM_Rapid.corr()
sns.heatmap(cor_Data, annot=True, cmap=plt.cm.Reds)
plt.show()
```

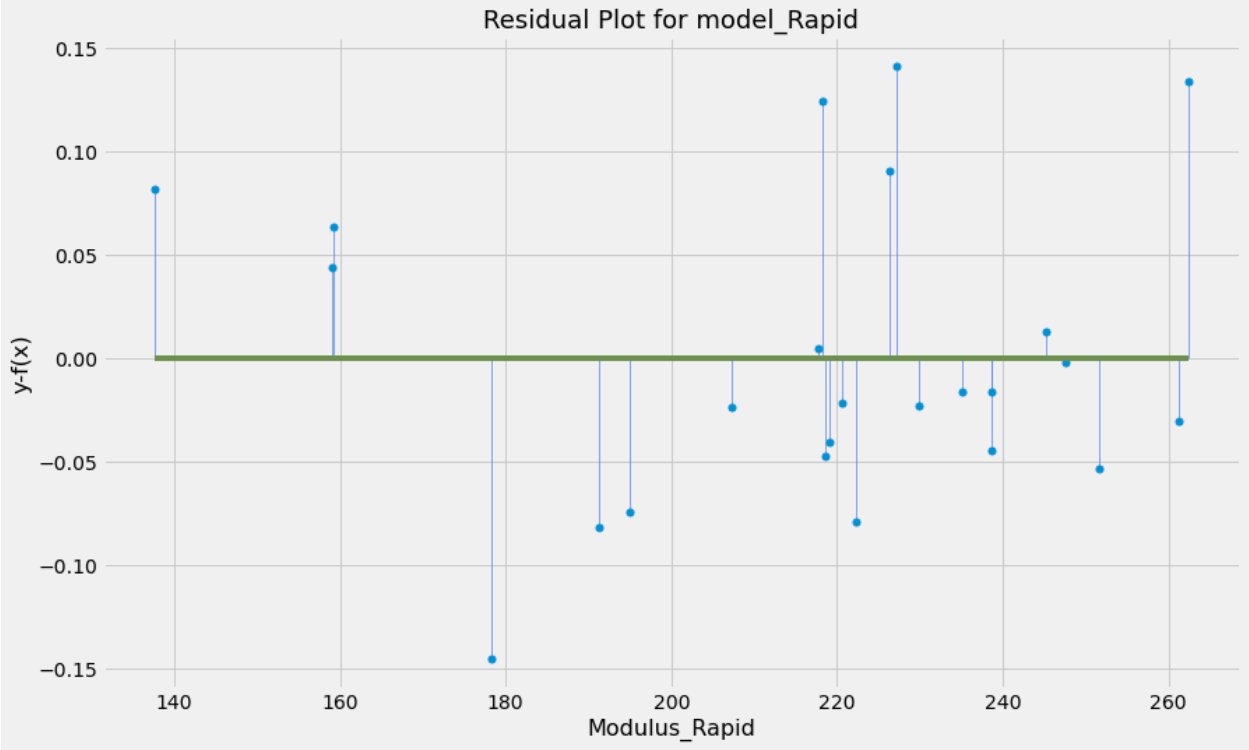



```
model_Rapid = linear_model.LinearRegression()

x_Rapid = CSM_Rapid_Test['Modulus_Rapid [Gpa]']
X_Rapid = CSM_Rapid_Test[['Modulus_Rapid [Gpa]']]
y_Rapid = CSM_Rapid_Test['Hardness_Rapid [Gpa]']

model_Rapid.fit(X_Rapid,y_Rapid)
model_Rapid.coef_
model_Rapid.intercept_

xfit_Rapid = np.linspace(125,275)
Xfit_Rapid = xfit_Rapid[:, np.newaxis]
yfit_Rapid = model_Rapid.predict(Xfit_Rapid)

plt.figure(figsize = (13,8))
plt.scatter(x_Rapid, y_Rapid, s=10, label='Original Datapoints')
plt.plot(xfit_Rapid, yfit_Rapid, c='orange', label='Linear Regression
```

```
Fitting');
plt.legend()
plt.title('Simple Linear Regression fitting of Modulus_Rapid vs.
Hardness_Rapid data', fontsize=18)
plt.ylabel('Hardness_Rapid [Gpa]', fontsize=16)
plt.xlabel('Modulus_Rapid [Gpa]', fontsize=16)


plt.figure(figsize=(13,8))
plt.title('Parity Plot of Original Modulus values against Modulus
values predicted by model', fontsize=18)
plt.scatter(CSM_Rapid_Test['Modulus_Rapid
[Gpa]'],model_Rapid.predict(CSM_Rapid_Test[['Modulus_Rapid [Gpa]']]),
s=10)
plt.xlabel('Original', fontsize=16)
plt.ylabel('Model', fontsize=16)
plt.plot(np.linspace(125, 275, 100),np.linspace(2.57,2.2,100), c='r')
plt.show()

plt.figure(figsize=(13,8))
plt.title('Residual Plot for model_Rapid', fontsize=18)
plt.xlabel('Modulus_Rapid', fontsize=16)
plt.ylabel('y-f(x)', fontsize=16)
markerline, stemline, baseline =
plt.stem(CSM_Rapid_Test['Modulus_Rapid [Gpa]'],

CSM_Rapid_Test['Hardness_Rapid [Gpa]']-
model_Rapid.predict(CSM_Rapid_Test[['Modulus_Rapid [Gpa]']]),
use_line_collection=True)
plt.setp(stemline, linewidth = 0.7, color='#5284F2')
plt.setp(markerline, markersize = 5)
plt.show()

print(f"R^2 SCORE = {r2_score(y_Rapid,
model_Rapid.intercept_+model_Rapid.coef_*CSM_Rapid_Test['Modulus_Rapid
[Gpa]'])}")
print('beta_0, beta_i: ', model_Rapid.intercept_, model_Rapid.coef_)
print(f"Mean Squared Error = {mean_squared_error(y_Rapid,
model_Rapid.predict(CSM_Rapid_Test[['Modulus_Rapid [Gpa]']]))}")
```

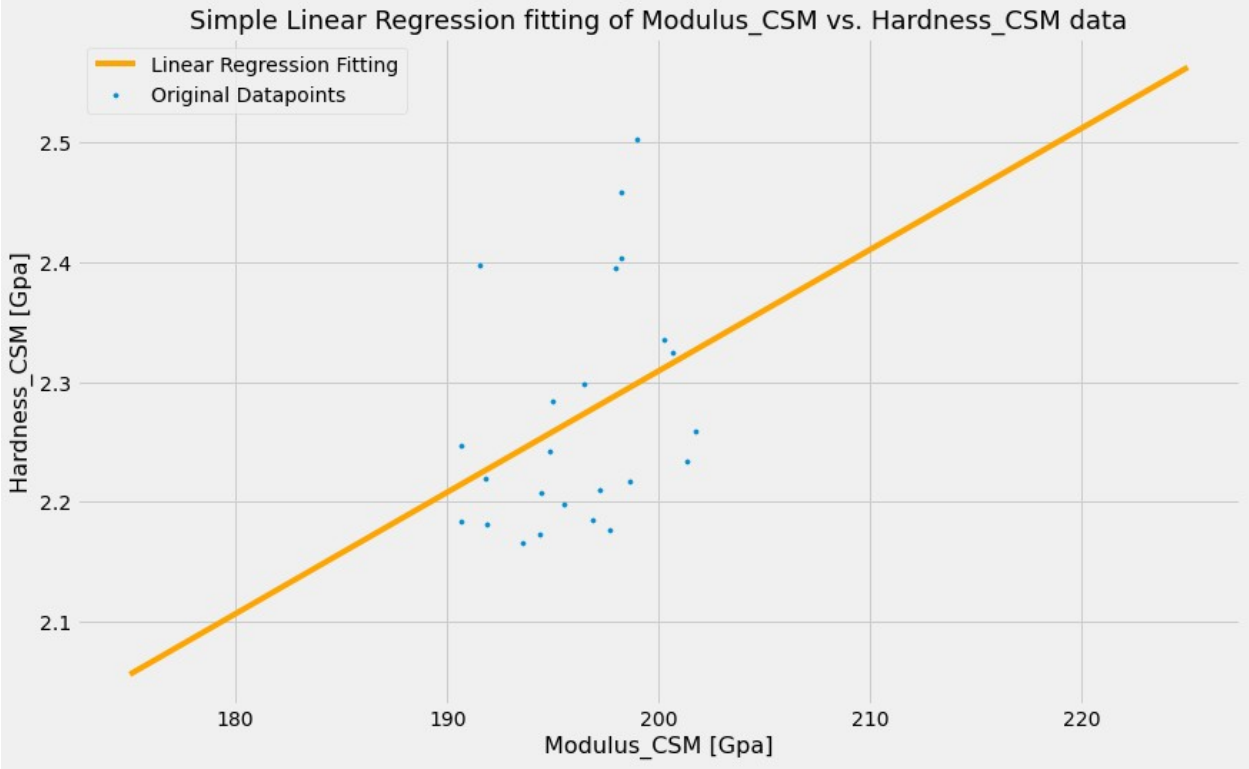

Simple Linear Regression fitting of Modulus_Rapid vs. Hardness_Rapid data
Linear Regression Fitting
Original Datapoints
Hardness_Rapid [Gpa]
Modulus_Rapid [Gpa]
2.6
2.5
2.4
2.3
2.2
120
140
160
180
200
220
240
260
280

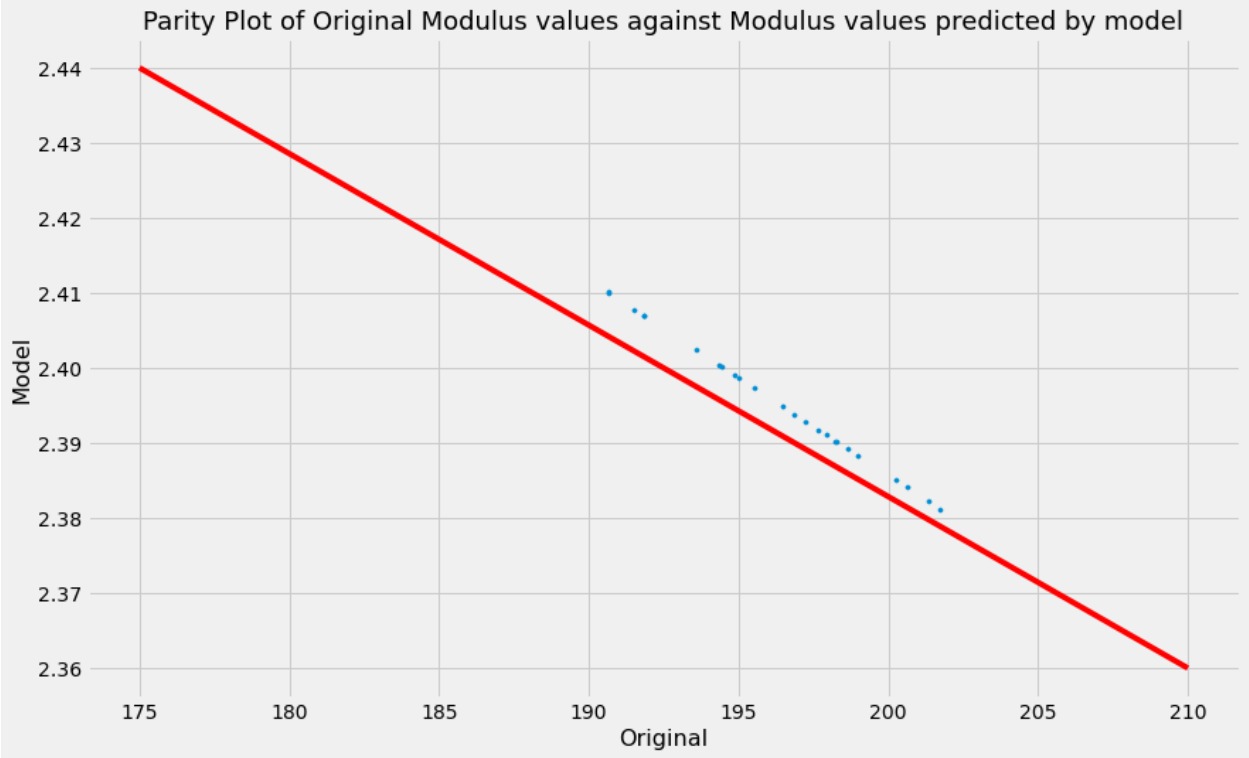

Parity Plot of Original Modulus values against Modulus values predicted by model
Model
Original
2.55
2.50
2.45
2.40
2.35
2.30
2.25
2.20
120
140
160
180
200
220
240
260
280

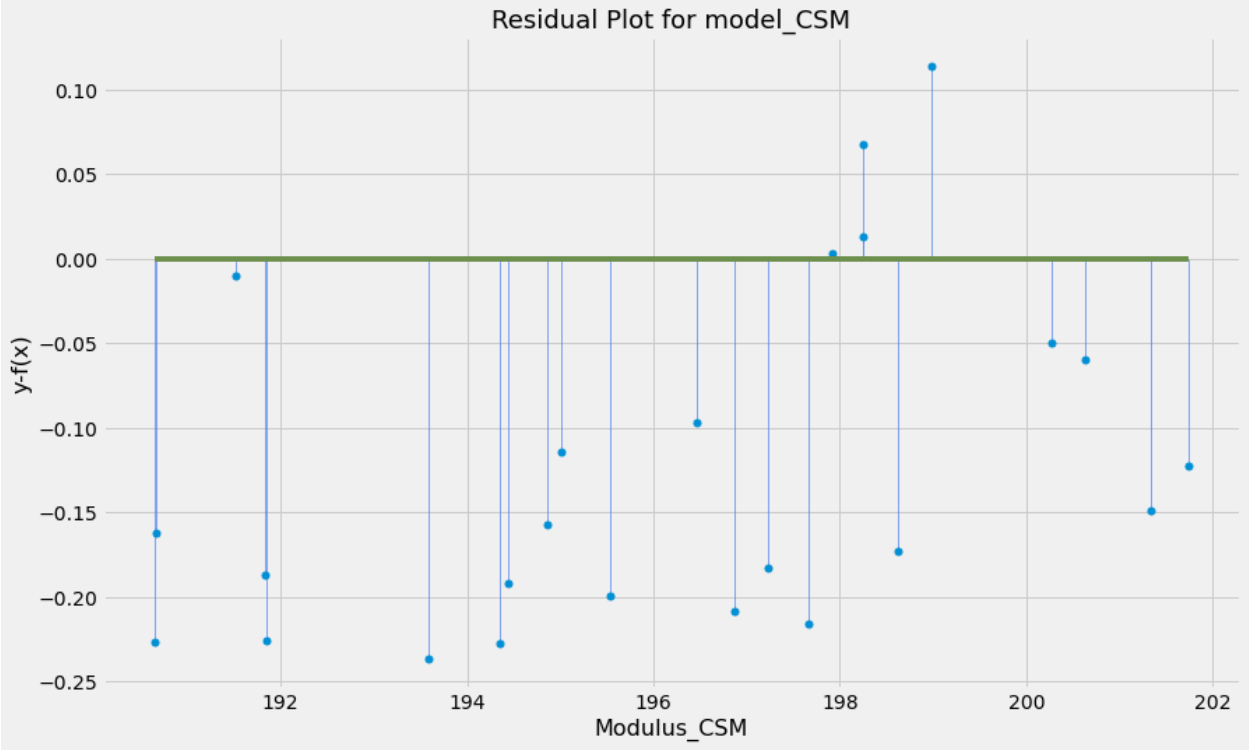



```
R^2 SCORE = 0.5683983099345535
beta_0, beta_i:  2.907008276184499 [-0.0026065]
Mean Squared Error = 0.005232607618535584

model_CSM = linear_model.LinearRegression()

x_CSM = CSM_Rapid_Test['Modulus_CSM [Gpa]']
X_CSM = CSM_Rapid_Test[['Modulus_CSM [Gpa]']]
y_CSM = CSM_Rapid_Test['Hardness_CSM [Gpa]']

model_CSM.fit(X_CSM,y_CSM)
model_CSM.coef_
model_CSM.intercept_

xfit_CSM = np.linspace(175,225)
Xfit_CSM = xfit_CSM[:, np.newaxis]
yfit_CSM = model_CSM.predict(Xfit_CSM)

plt.figure(figsize = (13,8))
plt.scatter(x_CSM, y_CSM, s=10, label='Original Datapoints')
plt.plot(xfit_CSM, yfit_CSM, c='orange', label='Linear Regression
Fitting');
plt.legend()
plt.title('Simple Linear Regression fitting of Modulus_CSM vs.
Hardness_CSM data',fontsize=18)
plt.ylabel('Hardness_CSM [Gpa]', fontsize=16)
plt.xlabel('Modulus_CSM [Gpa]', fontsize=16)
```

```
plt.figure(figsize=(13,8))
plt.title('Parity Plot of Original Modulus values against Modulus
values predicted by model', fontsize=18)
plt.scatter(CSM_Rapid_Test['Modulus_CSM
[Gpa]'],model_Rapid.predict(CSM_Rapid_Test[['Modulus_CSM [Gpa]']]),
s=10)
plt.xlabel('Original',fontsize=16)
plt.ylabel('Model',fontsize=16)
plt.plot(np.linspace(175, 210, 100),np.linspace(2.44,2.36,100), c='r')
plt.show()

plt.figure(figsize=(13,8))
plt.title('Residual Plot for model_CSM',fontsize=18)
plt.xlabel('Modulus_CSM', fontsize=16)
plt.ylabel('y-f(x)', fontsize=16)
markerline, stemline, baseline = plt.stem(CSM_Rapid_Test['Modulus_CSM
[Gpa]'],
                                          CSM_Rapid_Test['Hardness_CSM
[Gpa]']-model_Rapid.predict(CSM_Rapid_Test[['Modulus_CSM [Gpa]']]),
use_line_collection=True)
plt.setp(stemline, linewidth = 0.7, color='#5284F2')
plt.setp(markerline, markersize = 5)
plt.show()

print(f"R^2 SCORE = {r2_score(y_CSM,
model_CSM.intercept_+model_CSM.coef_*CSM_Rapid_Test['Modulus_CSM
[Gpa]'])}")
print('beta_0, beta_i: ', model_CSM.intercept_, model_CSM.coef_)
print(f"Mean Squared Error = {mean_squared_error(y_CSM,
model_CSM.predict(CSM_Rapid_Test[['Modulus_CSM [Gpa]']]))}")
```


Simple Linear Regression fitting of Modulus_CSM vs. Hardness_CSM data
Linear Regression Fitting
Original Datapoints
Hardness_CSM [Gpa]
Modulus_CSM [Gpa]
2.1
2.2
2.3
2.4
2.5
180
190
200
210
220


Parity Plot of Original Modulus values against Modulus values predicted by model
Model
Original
2.36
2.37
2.38
2.39
2.40
2.41
2.42
2.43
2.44
175
180
185
190
195
200
205
210



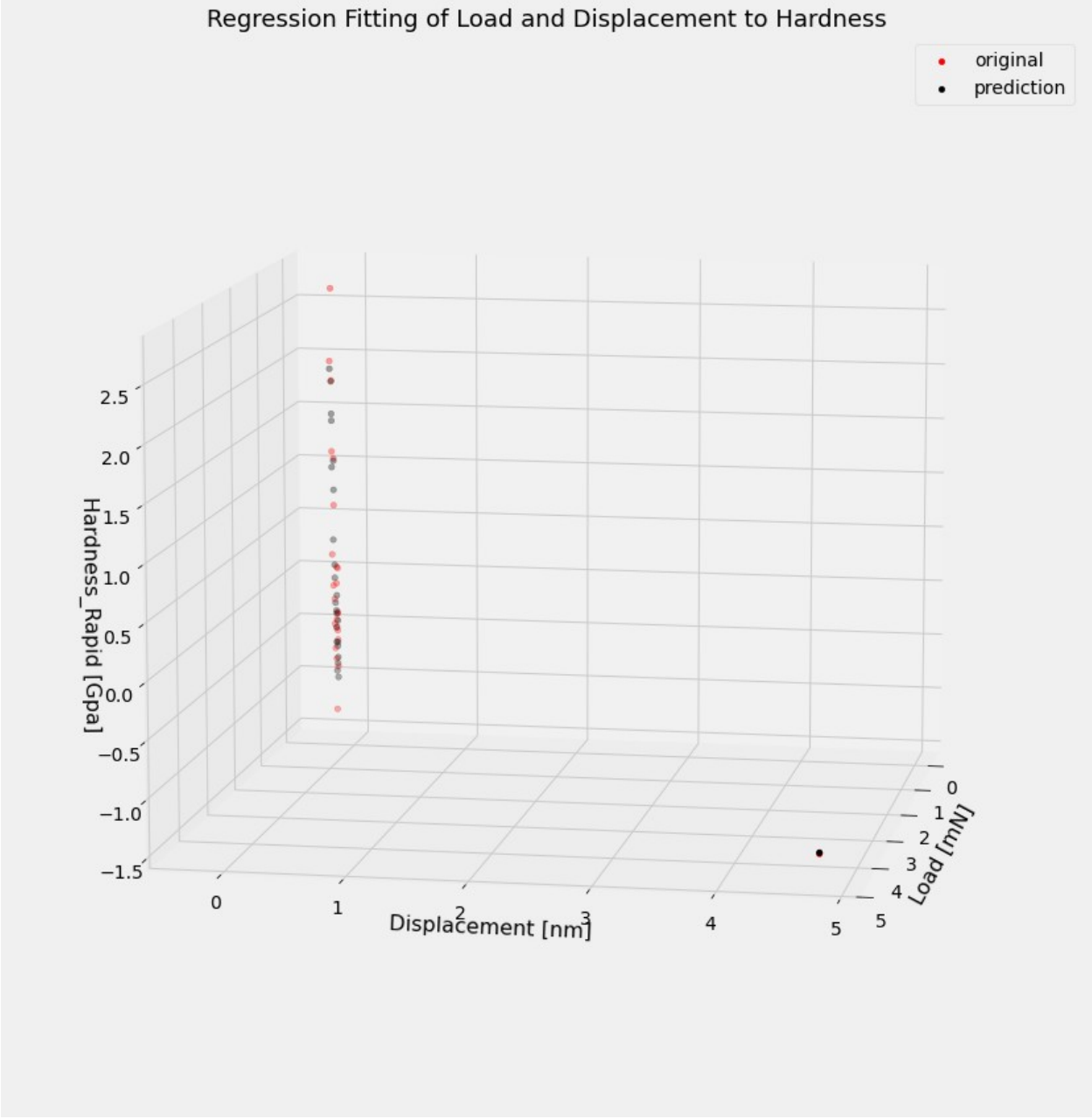

```
R^2 SCORE = 0.12161215077897891
beta_0, beta_i:  0.28161959905878087 [0.01013807]
Mean Squared Error = 0.008047394954601083
```

From the above two plots, since the R^2 value for $CSM = 12\%$, hence the correlation plot makes sense.

```
model = linear_model.LinearRegression()

Rapid_X = CSM_Rapid_Test[['Load [mN]', 'Displacement [nm]']]
Rapid_X = (Rapid_X - np.mean(Rapid_X, axis = 0))/ np.std(Rapid_X, axis
= 0)
Rapid_y = CSM_Rapid_Test['Hardness_Rapid [Gpa]']
Rapid_y = (Rapid_y - np.mean(Rapid_y, axis = 0))/ np.std(Rapid_y, axis
= 0)

model.fit(Rapid_X,Rapid_y)

#Step 5
plt.figure(figsize=(12,12))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(Rapid_X['Load [mN]'], Rapid_X['Displacement [nm]'],
Rapid_y, s=20,
                   c='r', label='original')
plot2=ax.scatter3D(Rapid_X['Load [mN]'], Rapid_X['Displacement [nm]'],
```

```
                    model.predict(Rapid_X[['Load [mN]', 'Displacement
[nm]']]), c='k', s=20,
                    label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(10,10)
ax.set_xlabel('Load [mN]', fontsize=16)
ax.set_ylabel('Displacement [nm]', fontsize=16)
ax.set_zlabel('Hardness_Rapid [Gpa]', fontsize=16)
ax.legend()
ax.set_title('Regression Fitting of Load and Displacement to
Hardness', fontsize=18)
plt.tight_layout()

r2 = r2_score(Rapid_y, model.predict(Rapid_X[['Load [mN]',
'Displacement [nm]']]))
MSE = mean_squared_error(Rapid_y, model.predict(Rapid_X[['Load [mN]',
'Displacement [nm]']]))
print(f'R^2 = {r2} \nMSE = {MSE}')
```

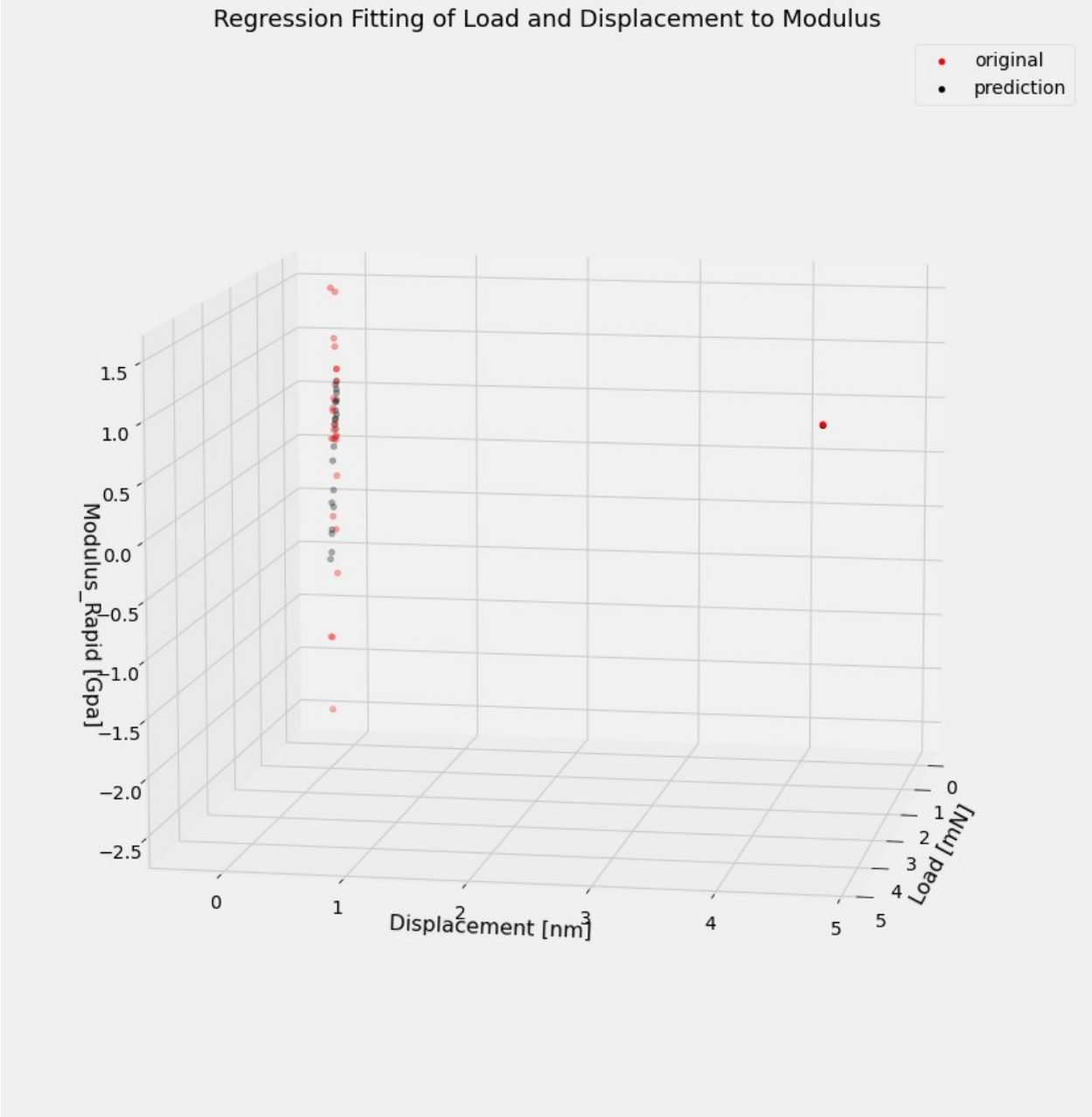
```
R^2 = 0.8657494845768515
MSE = 0.13425051542314845
```




```
model = linear_model.LinearRegression()

Rapid_X = CSM_Rapid_Test[['Load [mN]', 'Displacement [nm]']]
Rapid_X = (Rapid_X - np.mean(Rapid_X, axis = 0))/ np.std(Rapid_X, axis
= 0)
Rapid_y = CSM_Rapid_Test['Modulus_Rapid [Gpa]']
Rapid_y = (Rapid_y - np.mean(Rapid_y, axis = 0))/ np.std(Rapid_y, axis
= 0)


model.fit(Rapid_X,Rapid_y)
```

```
plt.figure(figsize=(12,12))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(Rapid_X['Load [mN]'], Rapid_X['Displacement [nm]'],
Rapid_y, s=20,
                   c='r', label='original')
plot2=ax.scatter3D(Rapid_X['Load [mN]'], Rapid_X['Displacement [nm]'],
                   model.predict(Rapid_X[['Load [mN]', 'Displacement
[nm]']]), c='k', s=20,
                   label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(10,10)
ax.set_xlabel('Load [mN]',fontsize=16)
ax.set_ylabel('Displacement [nm]',fontsize=16)
ax.set_zlabel('Modulus_Rapid [Gpa]',fontsize=16)
ax.legend()
ax.set_title('Regression Fitting of Load and Displacement to
Modulus',fontsize=18)
plt.tight_layout()

r2 = r2_score(Rapid_y, model.predict(Rapid_X[['Load [mN]',
'Displacement [nm]']]))
MSE = mean_squared_error(Rapid_y, model.predict(Rapid_X[['Load [mN]',
'Displacement [nm]']]))
print(f'R^2 = {r2} \nMSE = {MSE}')
```

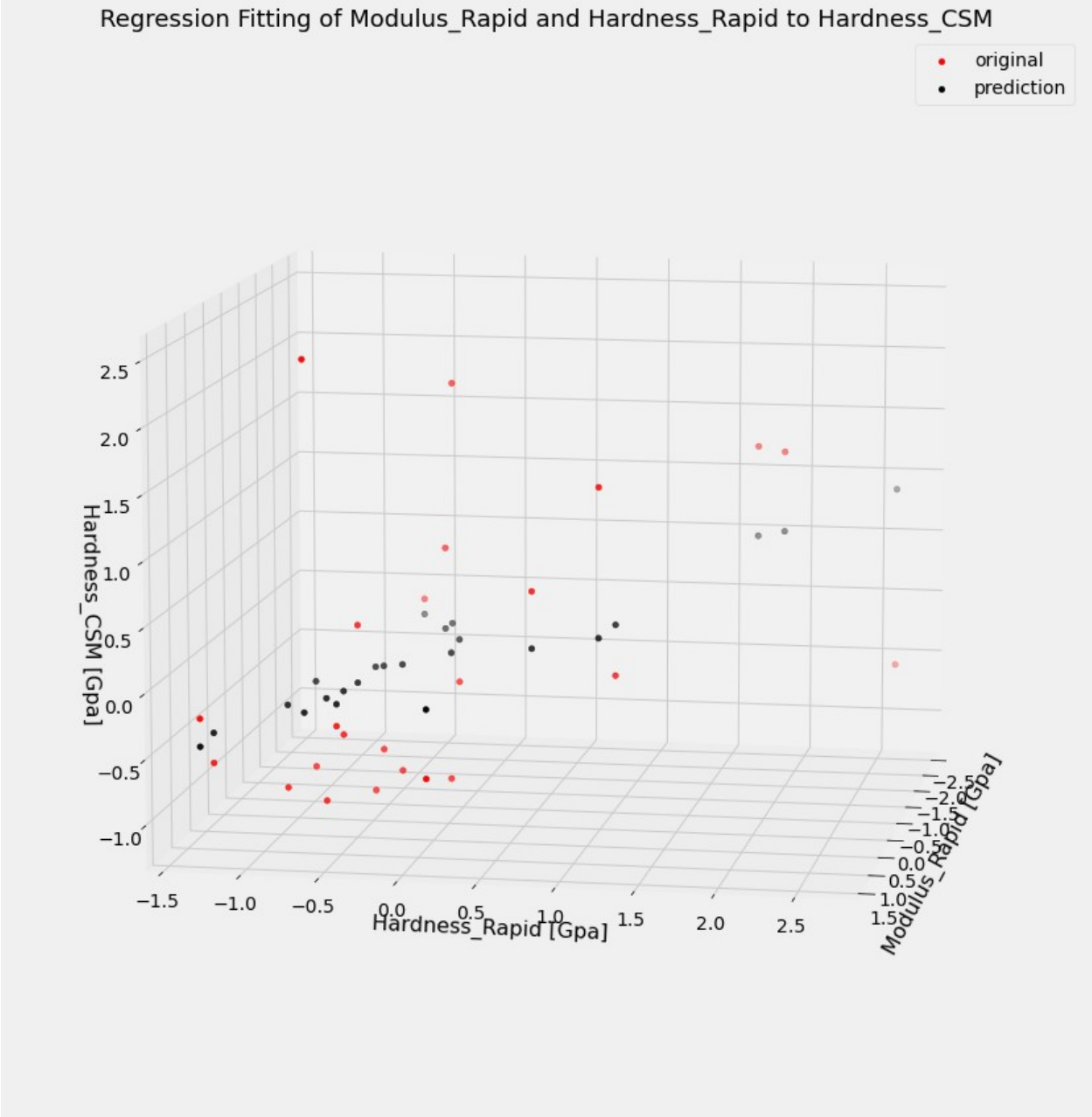
```
R^2 = 0.3104652462720473
MSE = 0.6895347537279526
```




```
model = linear_model.LinearRegression()

Feature_X = CSM_Rapid_Test[['Modulus_Rapid [Gpa]', 'Hardness_Rapid
[Gpa]']]
Feature_X = (Feature_X - np.mean(Feature_X, axis = 0))/
np.std(Feature_X, axis = 0)
Target_y = CSM_Rapid_Test['Hardness_CSM [Gpa]']
Target_y = (Target_y - np.mean(Target_y, axis = 0))/ np.std(Target_y,
axis = 0)

model.fit(Feature_X,Target_y)
```

```
plt.figure(figsize=(12,12))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(Feature_X['Modulus_Rapid [Gpa]'],
Feature_X['Hardness_Rapid [Gpa]'], Target_y, s=20,
                   c='r', label='original')
plot2=ax.scatter3D(Feature_X['Modulus_Rapid [Gpa]'],
Feature_X['Hardness_Rapid [Gpa]'],
                   model.predict(Feature_X[['Modulus_Rapid [Gpa]',
'Hardness_Rapid [Gpa]']]), c='k', s=20,
                   label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(10,10)
ax.set_xlabel('Modulus_Rapid [Gpa]',fontsize=16)
ax.set_ylabel('Hardness_Rapid [Gpa]',fontsize=16)
ax.set_zlabel('Hardness_CSM [Gpa]',fontsize=16)
ax.legend()
ax.set_title('Regression Fitting of Modulus_Rapid and Hardness_Rapid
to Hardness_CSM',fontsize=18)
plt.tight_layout()

r2 = r2_score(Target_y, model.predict(Feature_X[['Modulus_Rapid
[Gpa]', 'Hardness_Rapid [Gpa]']]))
MSE = mean_squared_error(Target_y,
model.predict(Feature_X[['Modulus_Rapid [Gpa]', 'Hardness_Rapid
[Gpa]']]))
print(f'R^2 = {r2} \nMSE = {MSE}')
```

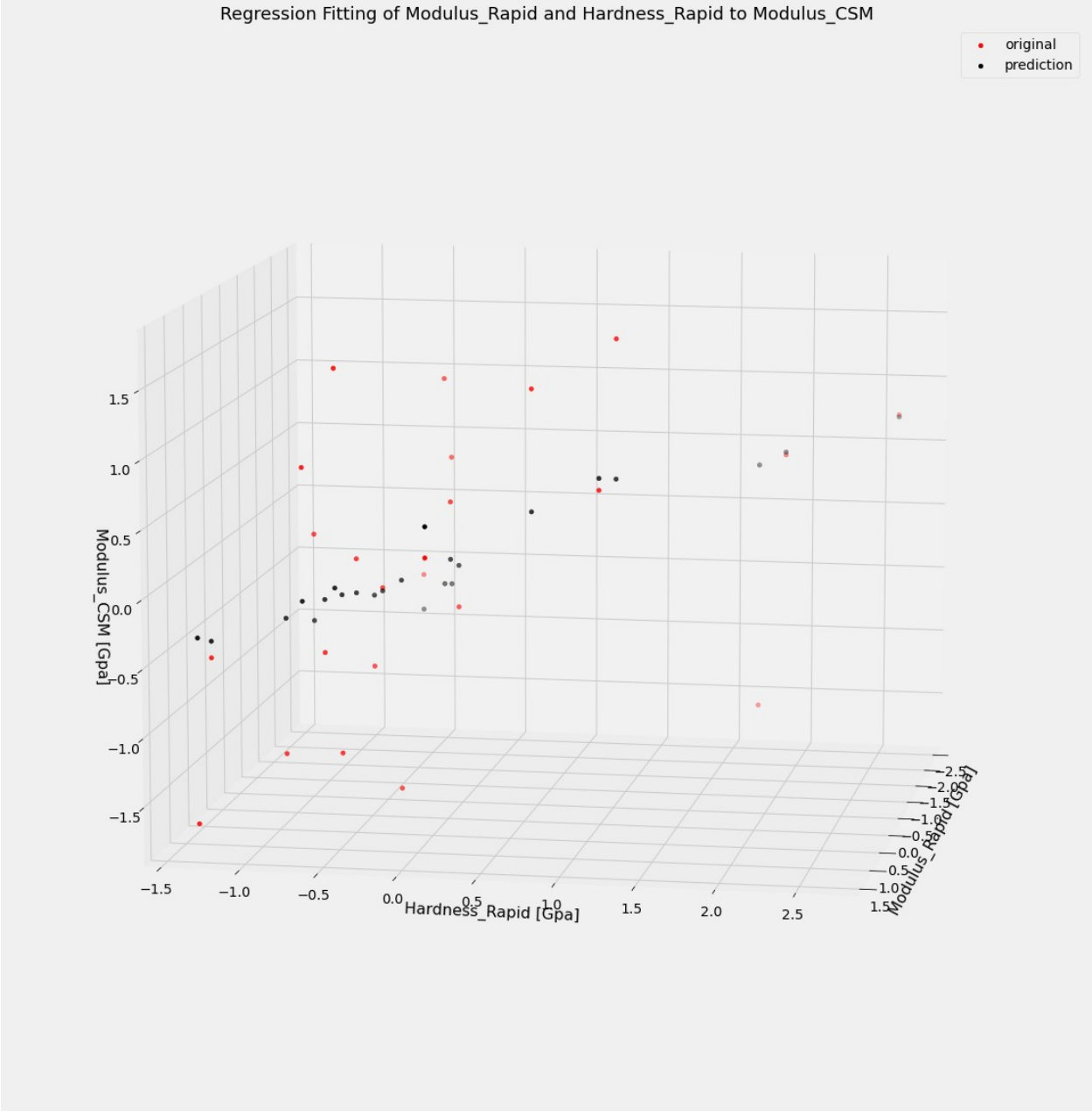
```
R^2 = 0.10481557710019407
MSE = 0.8951844228998059
```




```
model = linear_model.LinearRegression()

Feature_X = CSM_Rapid_Test[['Modulus_Rapid [Gpa]', 'Hardness_Rapid
[Gpa]']]
Feature_X = (Feature_X - np.mean(Feature_X, axis = 0))/
np.std(Feature_X, axis = 0)
Target_y = CSM_Rapid_Test['Modulus_CSM [Gpa]']
Target_y = (Target_y - np.mean(Target_y, axis = 0))/ np.std(Target_y,
axis = 0)

model.fit(Feature_X,Target_y)
```

```
plt.figure(figsize=(18,16))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(Feature_X['Modulus_Rapid [Gpa]'],
Feature_X['Hardness_Rapid [Gpa]'], Target_y, s=20,
                   c='r', label='original')
plot2=ax.scatter3D(Feature_X['Modulus_Rapid [Gpa]'],
Feature_X['Hardness_Rapid [Gpa]'],
                   model.predict(Feature_X[['Modulus_Rapid [Gpa]',
'Hardness_Rapid [Gpa]']]), c='k', s=20,
                   label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(10,10)
ax.set_xlabel('Modulus_Rapid [Gpa]', fontsize=16)
ax.set_ylabel('Hardness_Rapid [Gpa]', fontsize=16)
ax.set_zlabel('Modulus_CSM [Gpa]', fontsize=16)
ax.legend()
ax.set_title('Regression Fitting of Modulus_Rapid and Hardness_Rapid
to Modulus_CSM', fontsize=18)
plt.tight_layout()

r2 = r2_score(Target_y, model.predict(Feature_X[['Modulus_Rapid
[Gpa]', 'Hardness_Rapid [Gpa]']]))
MSE = mean_squared_error(Target_y,
model.predict(Feature_X[['Modulus_Rapid [Gpa]', 'Hardness_Rapid
[Gpa]']]))
print(f'R^2 = {r2} \nMSE = {MSE}')
```

```
R^2 = 0.16012586129224005
MSE = 0.83987413870776
```

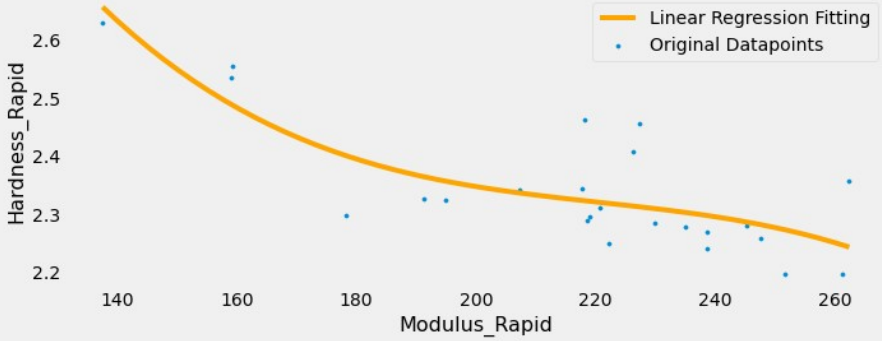



```
def make_model_1(n):
    poly_model = make_pipeline(PolynomialFeatures(n),
                               LinearRegression())
    x = CSM_Rapid_Test['Modulus_Rapid [Gpa]']
    X = CSM_Rapid_Test[['Modulus_Rapid [Gpa]']]
    y = CSM_Rapid_Test['Hardness_Rapid [Gpa]']
    poly_model.fit(X,y)

    xfit = np.linspace(x.min(),x.max())
    Xfit = xfit[:, np.newaxis] #this changes xfit from a horizontal
array to a vertical array!
```

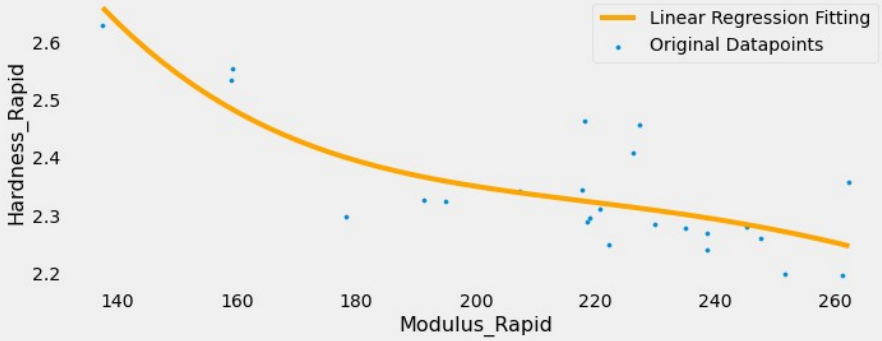
```
    yfit = poly_model.predict(Xfit)
    r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Modulus_Rapid
[Gpa]']]))

    return x, y, xfit, yfit, r2

plt.figure(figsize=(10,30))
for i in range(3,10):
    x, y, xfit, yfit, r2 = make_model_1(i)
    plt.subplot(7,1,i-2)
    plt.scatter(x, y, s=10, label='Original Datapoints')
    plt.plot(xfit, yfit, c='orange', label='Linear Regression
Fitting');
    plt.legend()
    plt.title(f'Simple Linear Regression fitting of Modulus_Rapid vs.
Hardness_Rapid data | R^2 = {r2} | Degree = {i}', fontsize=16)
    plt.ylabel('Hardness_Rapid',fontsize=16)
    plt.xlabel('Modulus_Rapid',fontsize=16)
    plt.grid()
plt.suptitle("For X and Y", y = 1.005, fontsize=20)
plt.tight_layout()
```

For X and Y

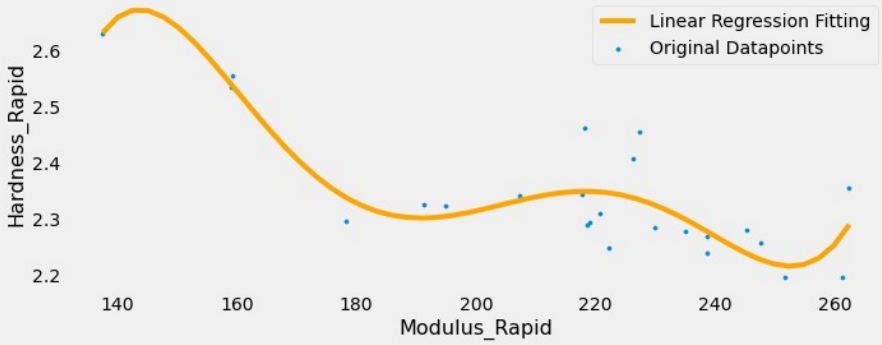
Simple Linear Regression fitting of Modulus_Rapid vs. Hardness_Rapid data | R^2 = 0.6431239282875112 | Degree = 3



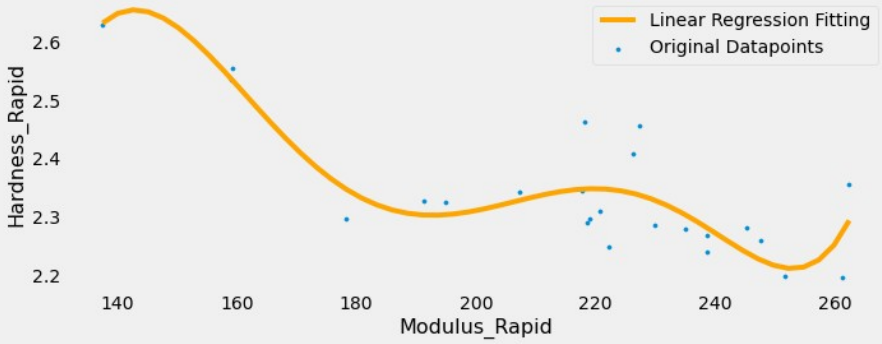
Simple Linear Regression fitting of Modulus_Rapid vs. Hardness_Rapid data | R^2 = 0.6435252740652755 | Degree = 4



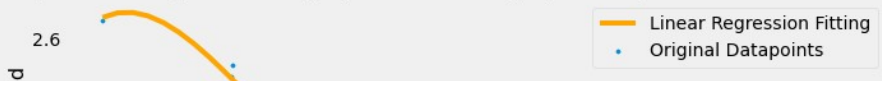
Simple Linear Regression fitting of Modulus_Rapid vs. Hardness_Rapid data | R^2 = 0.7597519248071388 | Degree = 5



Simple Linear Regression fitting of Modulus_Rapid vs. Hardness_Rapid data | R^2 = 0.7573945176353561 | Degree = 6



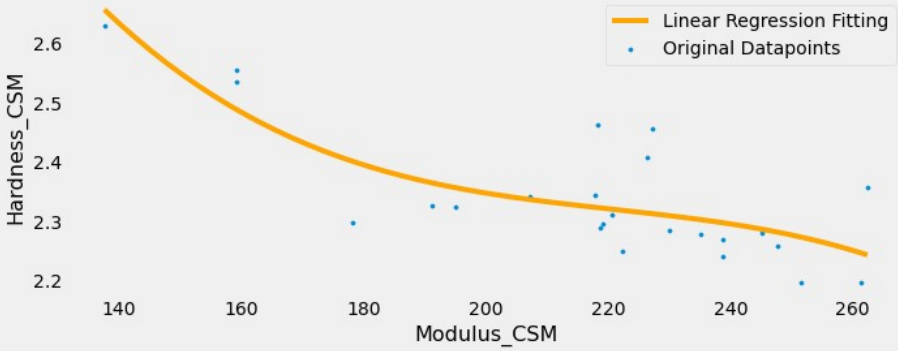
Simple Linear Regression fitting of Modulus_Rapid vs. Hardness_Rapid data | R^2 = 0.7541318156945911 | Degree = 7



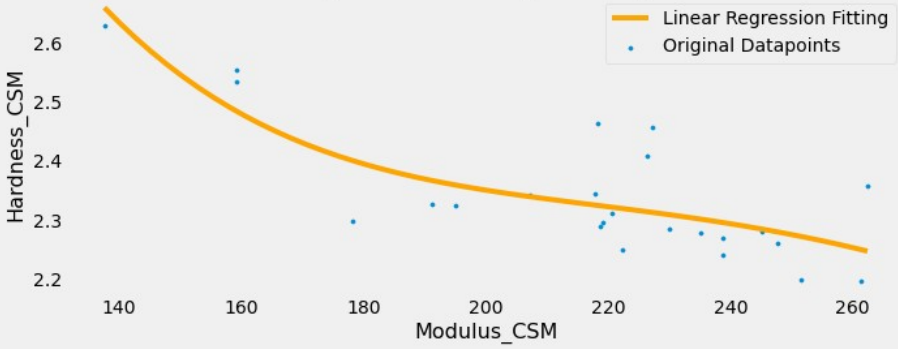
```
def make_model_2(n):
    poly_model = make_pipeline(PolynomialFeatures(n),
                               LinearRegression())
    x = CSM_Rapid_Test['Modulus_CSM [Gpa]']
    X = CSM_Rapid_Test[['Modulus_CSM [Gpa]']]
    y = CSM_Rapid_Test['Hardness_CSM [Gpa]']
    poly_model.fit(X,y)

    xfit = np.linspace(x.min(),x.max())
    Xfit = xfit[:, np.newaxis] #this changes xfit from a horizontal
array to a vertical array!
    yfit = poly_model.predict(Xfit)
    r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Modulus_CSM
[Gpa]']]))

    return x, y, xfit, yfit, r2

plt.figure(figsize=(10,30))
for i in range(3,10):
    x, y, xfit, yfit, r2 = make_model_1(i)
    plt.subplot(7,1,i-2)
    plt.scatter(x, y, s=10, label='Original Datapoints')
    plt.plot(xfit, yfit, c='orange', label='Linear Regression
Fitting');
    plt.legend()
    plt.title(f'Simple Linear Regression fitting of Modulus_CSM vs.
Hardness_CSM data | R^2 = {r2} | Degree = {i}',fontsize=16)
    plt.ylabel('Hardness_CSM',fontsize=16)
    plt.xlabel('Modulus_CSM',fontsize=16)
    plt.grid()
plt.suptitle("For X and Y", y = 1.005, fontsize=20)
plt.tight_layout()
```

For X and Y

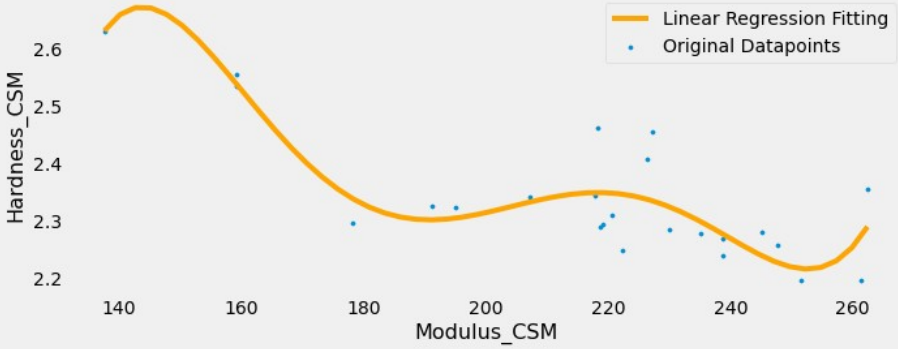
Simple Linear Regression fitting of Modulus_CSM vs. Hardness_CSM data | R^2 = 0.6431239282875112 | Degree = 3



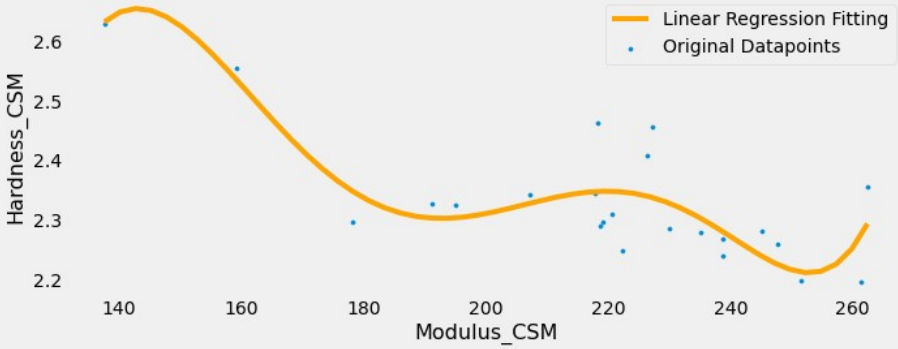
Simple Linear Regression fitting of Modulus_CSM vs. Hardness_CSM data | R^2 = 0.6435252740652755 | Degree = 4



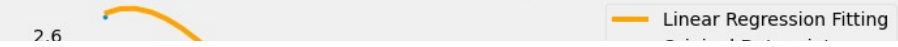
Simple Linear Regression fitting of Modulus_CSM vs. Hardness_CSM data | R^2 = 0.7597519248071388 | Degree = 5



Simple Linear Regression fitting of Modulus_CSM vs. Hardness_CSM data | R^2 = 0.7573945176353561 | Degree = 6



Simple Linear Regression fitting of Modulus_CSM vs. Hardness_CSM data | R^2 = 0.7541318156945911 | Degree = 7

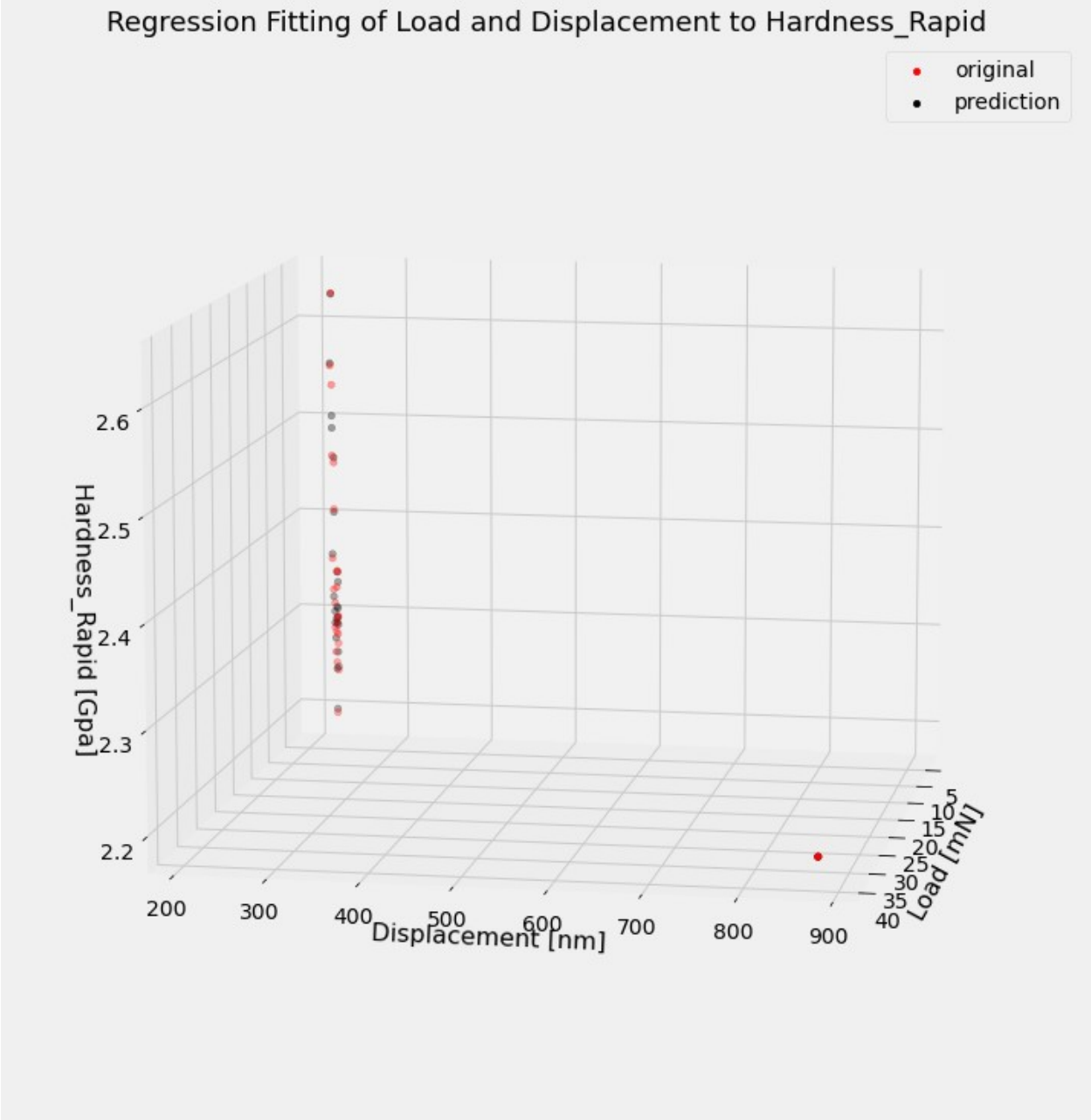


```
def make_model_3(n):
    poly_model = make_pipeline(PolynomialFeatures(n),
                               LinearRegression())
    X = CSM_Rapid_Test[['Load [mN]', 'Displacement [nm]']]
    y = CSM_Rapid_Test['Hardness_Rapid [Gpa]']

    # Initial plotting
    poly_model.fit(X,y)

    r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Load [mN]',
'Displacement [nm]']]))
    MSE = mean_squared_error(y,
poly_model.predict(CSM_Rapid_Test[['Load [mN]', 'Displacement
[nm]']]))

    return r2, MSE

for i in range(1, 30):
    r2, MSE = make_model_3(i)
    print(f'Degree = {i} | r2 = {r2} | MSE = {MSE}')
```

```
Degree = 1 | r2 = 0.8657494845768505 | MSE = 0.0016276124166450304
Degree = 2 | r2 = 0.9618520094554166 | MSE = 0.0004624946346367218 4
Degree = 3 | r2 = 0.9805410918725436 | MSE = 0.0002359138837029869 3
Degree = 4 | r2 = 0.9821304245112428 | MSE = 0.0002166452981875243 5
Degree = 5 | r2 = 0.9820563866441953 | MSE = 0.0002175429107689621 6
Degree = 6 | r2 = 0.9810518457785341 | MSE = 0.0002297216586927524 7
Degree = 7 | r2 = 0.980992316333559 | MSE = 0.0002304433755777274
Degree = 8 | r2 = 0.9766263433368817 | MSE = 0.0002833751042770984
Degree = 9 | r2 = 0.9614129549627513 | MSE = 0.0004678175977672059 4
Degree = 10 | r2 = 0.9611168560858077 | MSE = 0.0004714074104926848
Degree = 11 | r2 = 0.959863013778665 | MSE = 0.0004866086132678685
Degree = 12 | r2 = 0.9595723775491543 | MSE = 0.0004901321985173376
Degree = 13 | r2 = 0.9587339841273945 | MSE = 0.0005002966253650742
Degree = 14 | r2 = 0.9582423810761225 | MSE = 0.0005062566712374432
Degree = 15 | r2 = 0.9272465764148327 | MSE = 0.000882040379565165
Degree = 16 | r2 = 0.9256544488387073 | MSE = 0.0009013428500519017
Degree = 17 | r2 = 0.9238498127696803 | MSE = 0.0009232217088720403
Degree = 18 | r2 = 0.8750777652916633 | MSE = 0.001514519178458762
Degree = 19 | r2 = 0.8696871381986118 | MSE = 0.0015798735017736416
Degree = 20 | r2 = 0.5804688653358464 | MSE = 0.005086267876114318
Degree = 21 | r2 = 0.5768610747323656 | MSE = 0.005130007632080037
Degree = 22 | r2 = -4.060969855642833 | MSE = 0.061357659233910754
Degree = 23 | r2 = -3.9331376371785733 | MSE = 0.059807860297467846
Degree = 24 | r2 = -3.3874210460660317 | MSE = 0.053191758326727755
Degree = 25 | r2 = 0.07390758655760177 | MSE = 0.011227662749216502
Degree = 26 | r2 = 0.0739075865575991 | MSE = 0.011227662749216535
Degree = 27 | r2 = 0.0739075865575991 | MSE = 0.011227662749216535
Degree = 28 | r2 = 0.0739075865575991 | MSE = 0.011227662749216535
Degree = 29 | r2 = 0.0739075865575991 | MSE = 0.011227662749216535
```

From the above MSE values, we see that the MSE for degree = 4 is the least, hence plotting the same.

```
poly_model = make_pipeline(PolynomialFeatures(4), LinearRegression())

X = CSM_Rapid_Test[['Load [mN]', 'Displacement [nm]']]
y = CSM_Rapid_Test['Hardness_Rapid [Gpa]']

# Initial plotting
poly_model.fit(X,y)

plt.figure(figsize=(15,12))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(CSM_Rapid_Test['Load [mN]'],
CSM_Rapid_Test['Displacement [nm]'], CSM_Rapid_Test['Hardness_Rapid
[Gpa]'], s=20,
                   c='r', label='original')
plot2=ax.scatter3D(CSM_Rapid_Test['Load [mN]'],
CSM_Rapid_Test['Displacement [nm]'],
                   poly_model.predict(CSM_Rapid_Test[['Load [mN]',
'Displacement [nm]']]), c='k', s=20,
                   label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(10,10)
ax.set_xlabel('Load [mN]', fontsize=16)
ax.set_ylabel('Displacement [nm]', fontsize=16)
ax.set_zlabel('Hardness_Rapid [Gpa]', fontsize=16)
ax.legend()
ax.set_title('Regression Fitting of Load and Displacement to
Hardness_Rapid', fontsize=18)

r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Load [mN]',
'Displacement [nm]']]))
MSE = mean_squared_error(y, poly_model.predict((CSM_Rapid_Test[['Load
[mN]', 'Displacement [nm]']])))
print(f'R^2 = {r2} \nMSE = {MSE}')

R^2 = 0.9821304245112428
MSE = 0.00021664529818752435
```




```
def make_model_4(n):
    poly_model = make_pipeline(PolynomialFeatures(n),
                               LinearRegression())
    X = CSM_Rapid_Test[['Load [mN]', 'Displacement [nm]']]
    y = CSM_Rapid_Test['Modulus_Rapid [Gpa]']

    # Initial plotting
    poly_model.fit(X,y)

    r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Load [mN]',
'Displacement [nm]']]))
```

```
    MSE = mean_squared_error(y,
poly_model.predict(CSM_Rapid_Test[['Load [mN]', 'Displacement
[nm]']]))

    return r2, MSE

for i in range(1, 30):
    r2, MSE = make_model_4(i)
    print(f'Degree = {i} | r2 = {r2} | MSE = {MSE}')
```

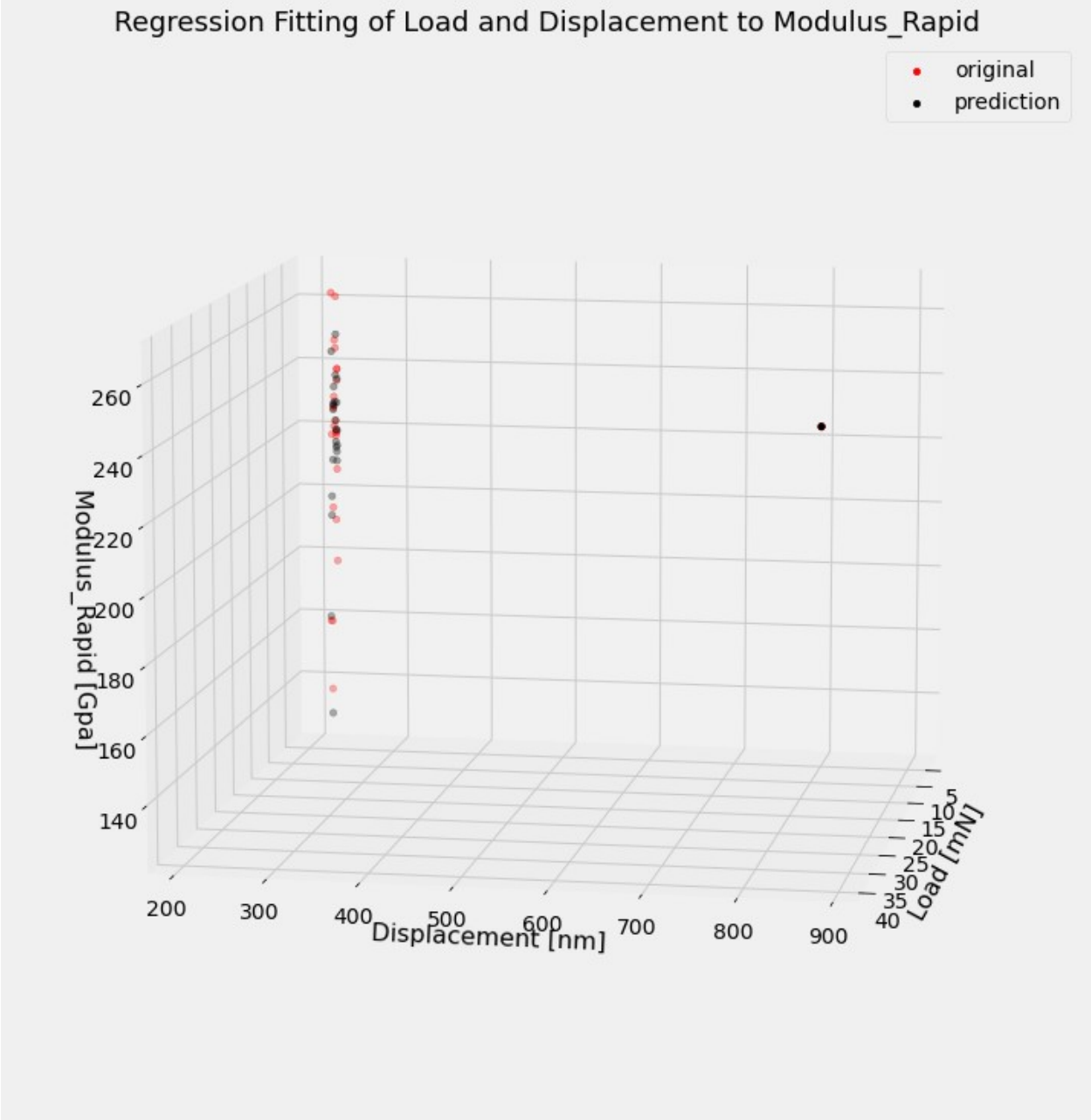
```
Degree = 1 | r2 = 0.31046524627204797 | MSE = 699.4044243254351
Degree = 2 | r2 = 0.4494659611525692 | MSE = 558.414119708837
Degree = 3 | r2 = 0.6427723587045735 | MSE = 362.34082685835864
Degree = 4 | r2 = 0.6736756243987927 | MSE = 330.9952266028436
Degree = 5 | r2 = 0.6725268351210842 | MSE = 332.1604591007025
Degree = 6 | r2 = 0.6512275670491194 | MSE = 353.76459470646637
Degree = 7 | r2 = 0.6503983860896809 | MSE = 354.60564416548544
Degree = 8 | r2 = 0.6021700269940142 | MSE = 403.52432092122336
Degree = 9 | r2 = 0.4550388161454837 | MSE = 552.7614975355589
Degree = 10 | r2 = 0.45313963354697717 | MSE = 554.6878641252281
Degree = 11 | r2 = 0.43930168481909815 | MSE = 568.7238826312447
Degree = 12 | r2 = 0.438054907355928 | MSE = 569.9885058705753
Degree = 13 | r2 = 0.43566127793900034 | MSE = 572.4163965538855
Degree = 14 | r2 = 0.4338103116860996 | MSE = 574.2938566522442
Degree = 15 | r2 = 0.38953370357473893 | MSE = 619.20421895057
Degree = 16 | r2 = 0.38881036707874606 | MSE = 619.9379089391285
Degree = 17 | r2 = 0.387966604908196 | MSE = 620.7937483177395
Degree = 18 | r2 = 0.2894225906923463 | MSE = 720.7482744104827
Degree = 19 | r2 = 0.285528661605598 | MSE = 724.6979393353572
Degree = 20 | r2 = 0.18828855093195906 | MSE = 823.3298984343625
Degree = 21 | r2 = 0.18652242674047392 | MSE = 825.1213020826262
Degree = 22 | r2 = -1.058695866205297 | MSE = 2088.1630539721123
Degree = 23 | r2 = -1.0405207272564008 | MSE = 2069.727764779129
Degree = 24 | r2 = -1.0333657144162602 | MSE = 2062.4703385080898
Degree = 25 | r2 = 0.051312398226889155 | MSE = 962.2666622610626
Degree = 26 | r2 = 0.05131239822688893 | MSE = 962.2666622610628
Degree = 27 | r2 = 0.0513123982268886 | MSE = 962.2666622610632
Degree = 28 | r2 = 0.0513123982268886 | MSE = 962.2666622610632
Degree = 29 | r2 = 0.0513123982268886 | MSE = 962.2666622610632
```

Using degree = 4 to visualize the regression.

```
poly_model = make_pipeline(PolynomialFeatures(4), LinearRegression())

X = CSM_Rapid_Test[['Load [mN]', 'Displacement [nm]']]
y = CSM_Rapid_Test['Modulus_Rapid [Gpa]']

# Initial plotting
poly_model.fit(X,y)
```

```
plt.figure(figsize=(15,12))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(CSM_Rapid_Test['Load [mN]'],
CSM_Rapid_Test['Displacement [nm]'], CSM_Rapid_Test['Modulus_Rapid
[Gpa]'], s=20,
                   c='r', label='original')
plot2=ax.scatter3D(CSM_Rapid_Test['Load [mN]'],
CSM_Rapid_Test['Displacement [nm]'],
                   poly_model.predict(CSM_Rapid_Test[['Load [mN]',
'Displacement [nm]']]), c='k', s=20,
                   label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(10,10)
ax.set_xlabel('Load [mN]', fontsize=16)
ax.set_ylabel('Displacement [nm]', fontsize=16)
ax.set_zlabel('Modulus_Rapid [Gpa]', fontsize=16)
ax.legend()
ax.set_title('Regression Fitting of Load and Displacement to
Modulus_Rapid', fontsize=18)

r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Load [mN]',
'Displacement [nm]']]))
MSE = mean_squared_error(y, poly_model.predict((CSM_Rapid_Test[['Load
[mN]', 'Displacement [nm]']])))
print(f'R^2 = {r2} \nMSE = {MSE}')
```

```
R^2 = 0.6736756243987927
MSE = 330.9952266028436
```




```
def make_model_5(n):
    poly_model = make_pipeline(PolynomialFeatures(n),
                               LinearRegression())
    X = CSM_Rapid_Test[['Modulus_Rapid [Gpa]', 'Hardness_Rapid
[Gpa]']]
    y = CSM_Rapid_Test['Hardness_CSM [Gpa]']

    # Initial plotting
    poly_model.fit(X,y)

    r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Modulus_Rapid
```

```
[Gpa]', 'Hardness_Rapid [Gpa]']]))
    MSE = mean_squared_error(y,
poly_model.predict(CSM_Rapid_Test[['Modulus_Rapid [Gpa]',
'Hardness_Rapid [Gpa]']]))

    return r2, MSE

for i in range(1, 30):
    r2, MSE = make_model_5(i)
    print(f'Degree = {i} | r2 = {r2} | MSE = {MSE}')
```

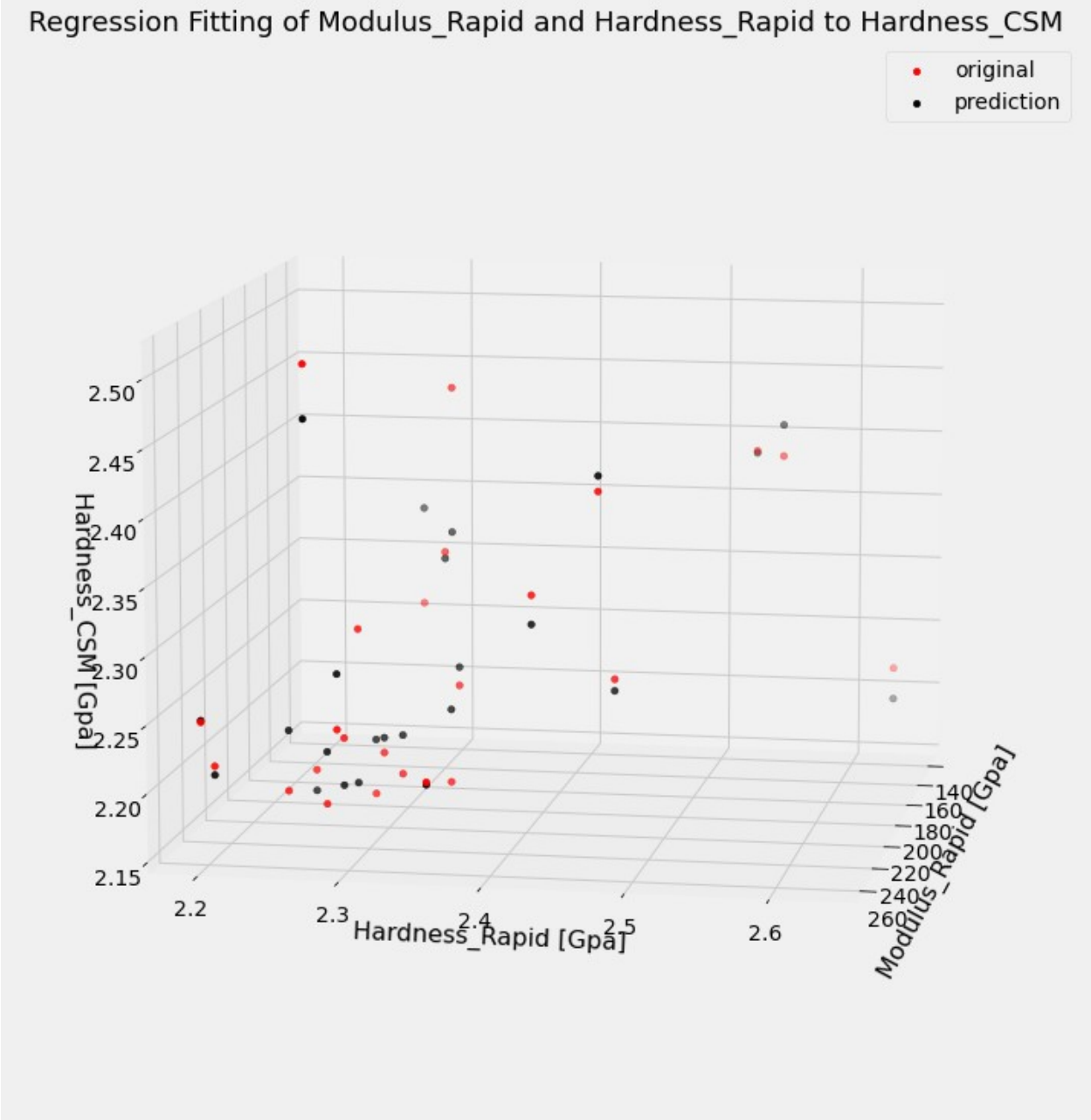
```
Degree = 1 | r2 = 0.10481557710019362 | MSE = 0.008201277618616887
Degree = 2 | r2 = 0.15140375813460805 | MSE = 0.00777445762863994
Degree = 3 | r2 = 0.4498762077448103 | MSE = 0.005039987101513837
Degree = 4 | r2 = 0.7042826493281744 | MSE = 0.002709229548080477
Degree = 5 | r2 = 0.7297569776408934 | MSE = 0.0024758451936436198
Degree = 6 | r2 = 0.7360473059463968 | MSE = 0.002418216030952693
Degree = 7 | r2 = 0.7445620909359637 | MSE = 0.0023402073952169854
Degree = 8 | r2 = 0.7565843286988403 | MSE = 0.002230065052512921
Degree = 9 | r2 = 0.7724332260312488 | MSE = 0.0020848645735423586
Degree = 10 | r2 = 0.7827839240628187 | MSE = 0.0019900361270995663
Degree = 11 | r2 = 0.7810333353194946 | MSE = 0.0020060742349047927
Degree = 12 | r2 = 0.7718062217229554 | MSE = 0.0020906089054016255
Degree = 13 | r2 = 0.7647651039939772 | MSE = 0.0021551164635801472
Degree = 14 | r2 = 0.7631273727603675 | MSE = 0.002170120621570301
Degree = 15 | r2 = 0.7664188395119931 | MSE = 0.0021399656815244447
Degree = 16 | r2 = 0.7742141807633682 | MSE = 0.002068548266186399
Degree = 17 | r2 = 0.7863352580700034 | MSE = 0.0019575003999753055
Degree = 18 | r2 = 0.8025963835717111 | MSE = 0.0018085232716661792
Degree = 19 | r2 = 0.6383960232672456 | MSE = 0.0033128532236682217
Degree = 20 | r2 = 0.626030370974128 | MSE = 0.0034261417760568223
Degree = 21 | r2 = 0.6238765091511171 | MSE = 0.0034458744906916802
Degree = 22 | r2 = 0.6329963557905744 | MSE = 0.0033623225518776474
Degree = 23 | r2 = 0.6503664348913165 | MSE = 0.0032031856887706537
Degree = 24 | r2 = 0.6443094718188406 | MSE = 0.003258676864010468
Degree = 25 | r2 = 0.6506290744933039 | MSE = 0.0032007795026994476
Degree = 26 | r2 = 0.6576660048770748 | MSE = 0.003136310307097006
Degree = 27 | r2 = 0.6657467684152628 | MSE = 0.003062277980962022
Degree = 28 | r2 = 0.6749099460409314 | MSE = 0.002978329063114043
Degree = 29 | r2 = 0.6849667329958465 | MSE = 0.0028861932979480586
```

In the above optimization, we see that the MSE is the minimum for degree = 17.

```
poly_model = make_pipeline(PolynomialFeatures(17), LinearRegression())

X = CSM_Rapid_Test[['Modulus_Rapid [Gpa]', 'Hardness_Rapid [Gpa]']]
y = CSM_Rapid_Test['Hardness_CSM [Gpa]']
# Initial plotting
poly_model.fit(X,y)
```

```
plt.figure(figsize=(15,12))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(CSM_Rapid_Test['Modulus_Rapid [Gpa]'],
CSM_Rapid_Test['Hardness_Rapid [Gpa]'], CSM_Rapid_Test['Hardness_CSM
[Gpa]'], s=20,
                   c='r', label='original')
plot2=ax.scatter3D(CSM_Rapid_Test['Modulus_Rapid [Gpa]'],
CSM_Rapid_Test['Hardness_Rapid [Gpa]'],
                   poly_model.predict(CSM_Rapid_Test[['Modulus_Rapid
[Gpa]', 'Hardness_Rapid [Gpa]']]), c='k', s=20,
                   label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(10,10)
ax.set_xlabel('Modulus_Rapid [Gpa]', fontsize=16)
ax.set_ylabel('Hardness_Rapid [Gpa]', fontsize=16)
ax.set_zlabel('Hardness_CSM [Gpa]', fontsize=16)
ax.legend()
ax.set_title('Regression Fitting of Modulus_Rapid and Hardness_Rapid
to Hardness_CSM', fontsize=18)

r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Modulus_Rapid
[Gpa]', 'Hardness_Rapid [Gpa]']]))
MSE = mean_squared_error(y,
poly_model.predict((CSM_Rapid_Test[['Modulus_Rapid [Gpa]',
'Hardness_Rapid [Gpa]']])))
print(f'R^2 = {r2} \nMSE = {MSE}')
```

```
R^2 = 0.7863352580700034
MSE = 0.0019575003999753055
```




```
def make_model_6(n):
    poly_model = make_pipeline(PolynomialFeatures(n),
                               LinearRegression())
    X = CSM_Rapid_Test[['Modulus_Rapid [Gpa]', 'Hardness_Rapid
[Gpa]']]
    y = CSM_Rapid_Test['Modulus_CSM [Gpa]']

    # Initial plotting
    poly_model.fit(X,y)

    r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Modulus_Rapid
```

```
[Gpa]', 'Hardness_Rapid [Gpa]']]))
    MSE = mean_squared_error(y,
poly_model.predict(CSM_Rapid_Test[['Modulus_Rapid [Gpa]',
'Hardness_Rapid [Gpa]']]))

    return r2, MSE

for i in range(1, 30):
    r2, MSE = make_model_6(i)
    print(f'Degree = {i} | r2 = {r2} | MSE = {MSE}')
```

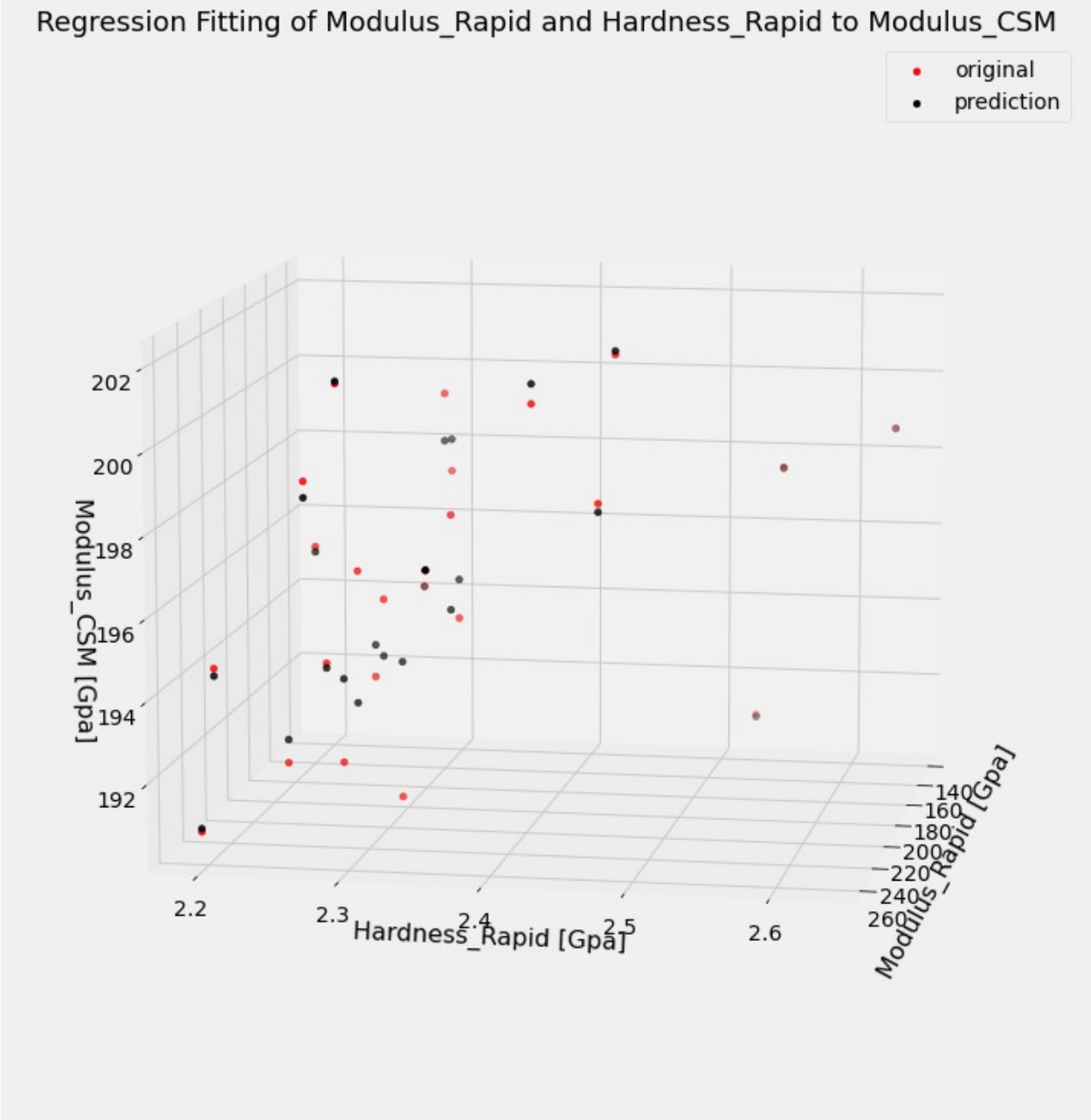
```
Degree = 1 | r2 = 0.16012586129224138 | MSE = 9.104366307059685
Degree = 2 | r2 = 0.2611390900285 | MSE = 8.00936719482519
Degree = 3 | r2 = 0.3965805567403332 | MSE = 6.541160627580201
Degree = 4 | r2 = 0.8129173420327228 | MSE = 2.0280051133055745
Degree = 5 | r2 = 0.8488272271359363 | MSE = 1.6387363729594195
Degree = 6 | r2 = 0.8502134458575122 | MSE = 1.6237095463894884
Degree = 7 | r2 = 0.8506043805136386 | MSE = 1.619471753906706
Degree = 8 | r2 = 0.851819359315411 | MSE = 1.606301194704029
Degree = 9 | r2 = 0.8370108847733628 | MSE = 1.7668273622164765
Degree = 10 | r2 = 0.8261303168629495 | MSE = 1.8847744108512376
Degree = 11 | r2 = 0.7981096279090979 | MSE = 2.18852303776403
Degree = 12 | r2 = 0.7610694601948291 | MSE = 2.5900442174308913
Degree = 13 | r2 = 0.7335740203881862 | MSE = 2.8880990618847826
Degree = 14 | r2 = 0.7182278815793692 | MSE = 3.0544535936833506
Degree = 15 | r2 = 0.7093753459775043 | MSE = 3.1504164566304973
Degree = 16 | r2 = 0.703095967925299 | MSE = 3.2184858914814742
Degree = 17 | r2 = 0.6977335600996146 | MSE = 3.2766152264411033
Degree = 18 | r2 = 0.6928143522795133 | MSE = 3.3299402043999016
Degree = 19 | r2 = 0.6684980700108871 | MSE = 3.593532486619779
Degree = 20 | r2 = 0.6695876895132804 | MSE = 3.5817208417221664
Degree = 21 | r2 = 0.6713767630765812 | MSE = 3.5623270060033665
Degree = 22 | r2 = 0.6733749853466507 | MSE = 3.540665965769852
Degree = 23 | r2 = 0.6753814553852131 | MSE = 3.5189155184429244
Degree = 24 | r2 = 0.6595421731052267 | MSE = 3.69061580217805
Degree = 25 | r2 = 0.6566407174873146 | MSE = 3.72206803240125
Degree = 26 | r2 = 0.6532336840934467 | MSE = 3.7590008043590615
Degree = 27 | r2 = 0.6496934666014036 | MSE = 3.797377312658046
Degree = 28 | r2 = 0.6463611391470482 | MSE = 3.833500260610941
Degree = 29 | r2 = 0.6435006173293545 | MSE = 3.8645087620724703
```

Using degree = 8, to plot the regression.

```
poly_model = make_pipeline(PolynomialFeatures(8), LinearRegression())

X = CSM_Rapid_Test[['Modulus_Rapid [Gpa]', 'Hardness_Rapid [Gpa]']]
y = CSM_Rapid_Test['Modulus_CSM [Gpa]']
# Initial plotting
poly_model.fit(X,y)
```

```
plt.figure(figsize=(15,12))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(CSM_Rapid_Test['Modulus_Rapid [Gpa]'],
CSM_Rapid_Test['Hardness_Rapid [Gpa]'], CSM_Rapid_Test['Modulus_CSM
[Gpa]'], s=20,
                   c='r', label='original')
plot2=ax.scatter3D(CSM_Rapid_Test['Modulus_Rapid [Gpa]'],
CSM_Rapid_Test['Hardness_Rapid [Gpa]'],
                   poly_model.predict(CSM_Rapid_Test[['Modulus_Rapid
[Gpa]', 'Hardness_Rapid [Gpa]']]), c='k', s=20,
                   label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(10,10)
ax.set_xlabel('Modulus_Rapid [Gpa]', fontsize=16)
ax.set_ylabel('Hardness_Rapid [Gpa]', fontsize=16)
ax.set_zlabel('Modulus_CSM [Gpa]', fontsize=16)
ax.legend()
ax.set_title('Regression Fitting of Modulus_Rapid and Hardness_Rapid
to Modulus_CSM', fontsize=18)

r2 = r2_score(y, poly_model.predict(CSM_Rapid_Test[['Modulus_Rapid
[Gpa]', 'Hardness_Rapid [Gpa]']]))
MSE = mean_squared_error(y,
poly_model.predict((CSM_Rapid_Test[['Modulus_Rapid [Gpa]',
'Hardness_Rapid [Gpa]']])))
print(f'R^2 = {r2} \nMSE = {MSE}')
```

```
R^2 = 0.851819359315411
MSE = 1.606301194704029
```



Prediction of Nanoindentation Hardness in as Built Condition

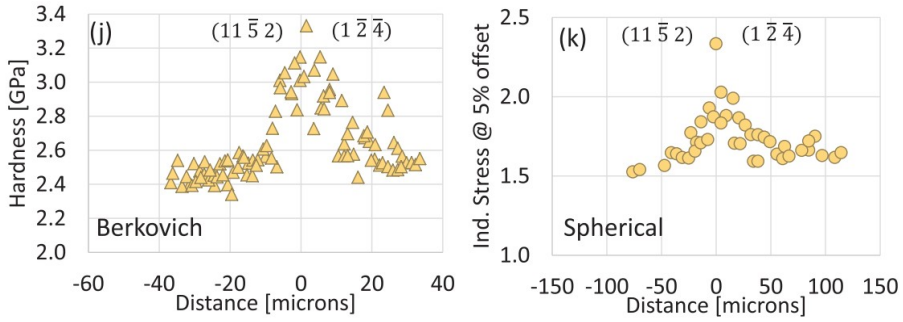
In this study, the authors reported hardness values and modulus values obtained by berkovich indenter after shock loading condition. The measurements were taken accorss the grain boundaries of GB1, GB2, and GB3. The authors took around 100 indentation across each grain boundary with berkovich indenter. For berkovich nanoindentation, measurements were taken within 80 microns, (half left from the grain boundary and half right from the grain boundary) (as shown in following cell). The author also took spherical nanoindentation measurement. For the

spherical nanoindentation, less measurements were taken (40 indentations accross grain boundary), however more with more spatial variation (within 200 microns).

The authors did not reported the nanoindentation hardness of asbuilt (additively manufactured) tantalum, they only reported the nanoindentation properties after shock loading, which includes berkovich nanohardness, modulus by berkovich method, 5% indentation stress by spherical method (not nanoindentation hardness) and modulus by spherical method. In the dataset, the as-built indentation hardness at 5% offset were found, however the indentation hardness of asbuilt Tantalum according to berkovich method was not reported, possibly due to avoid tip damage. The spherical indentater is wider, which could be more compatible since the as-built specimen after additive manufacturing has higher roughness, peaks and valleys.

That is why, our group has decided to predict the nano-hardness according to berkovich method for as-built condition. For this analysis, the corresponding nanoindenations from both berkovich and spherical were used for analysis, however, within the common region (80 microns). This process required extensive data cleaning, which was done in exce to avoid computational time and complexity. Linear and multi-linear algorithm have been employed, where the "As built Spherical indentation stress", "As built shperical modulus", "Shock loaded berkovich hardness", and "shock loaded berkovich modulus" data were were used to train data. From this training, models have been developed as shown below. The, the "As built spherical hardness" and "As built Shperical modulus" have been used to predict the "As-built berkovich hardness" and "As built berkovich modulus". The resutls have been comapred with literatures.

```
from IPython import display
display.Image("Author_result.png")
```



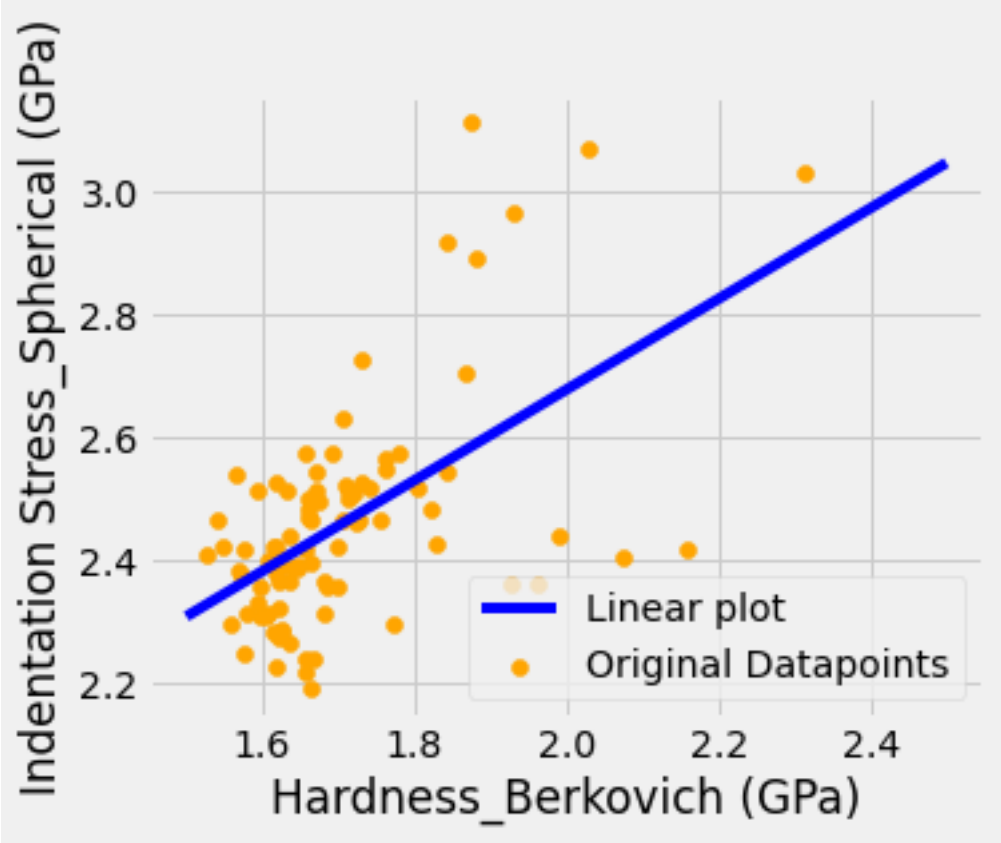

```
Given_data = pd.read_csv('Ind_Stress_Mod_to_target_Berk_Mod_Hardness
(1).csv');
Given_data.keys()

Index(['Test', 'Displacement Into Surface (mm)', 'Load On Sample
(mN)',
       'Modulus_Berk (GPa)', 'Hardness_Berk (GPa)', 'Condition_Berk',
       'Grain_Berk', 'Test_Berk', 'Distance_Berk', 'Modulus_Spherical
(GPa)',
       'Indentation Stress_Spherical (GPa)', 'Condition_Spherical',
       'Grain_Spherical', 'Test_Spherical', 'Distance_Spherical',
       'Modulus_As-built_Spherical (GPa)', 'Indentation Stress_As-
Built (GPa)',
```

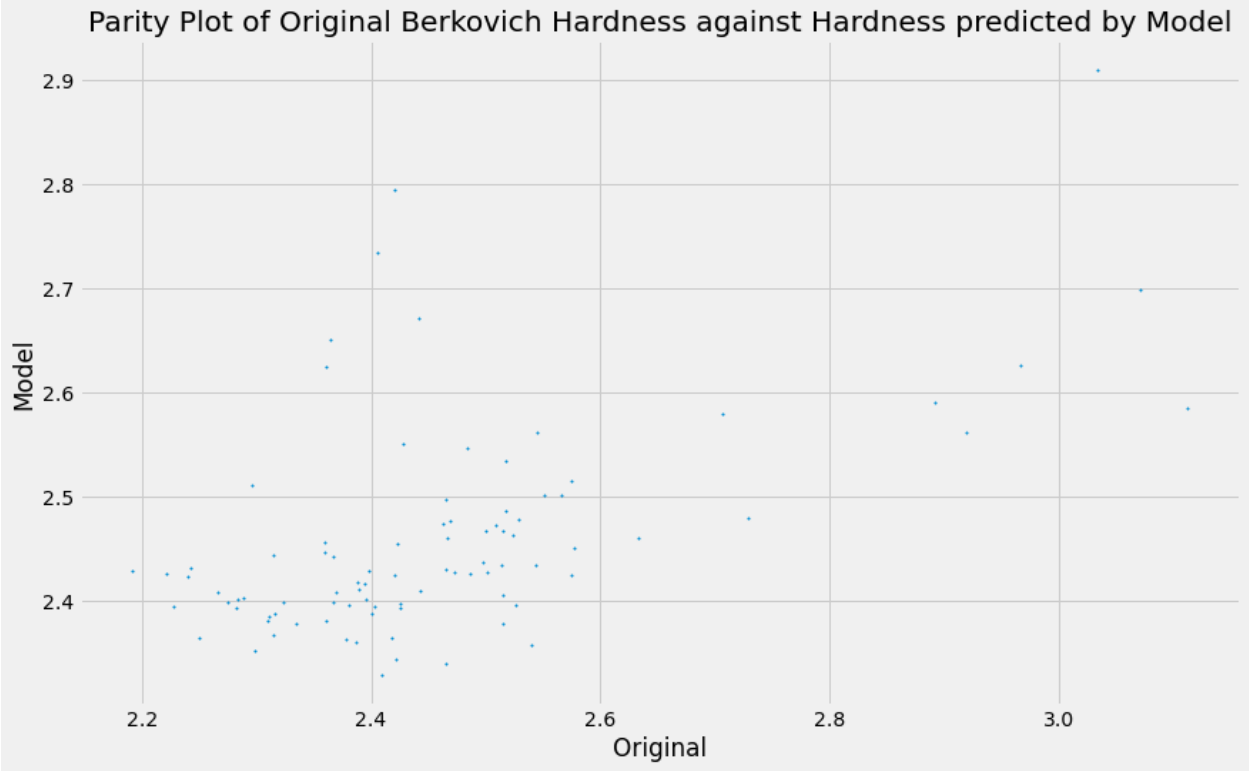
```
       'Condition_As-built', 'Grain_As-Built', 'Test_As-Built'],
      dtype='object')

cor = Given_data[['Modulus_Berk (GPa)','Hardness_Berk (GPa)',
'Modulus_Spherical (GPa)', 'Indentation Stress_Spherical (GPa)']]
plt.figure(figsize=(12,10))
cor_Data = cor.corr()
sns.heatmap(cor_Data, annot=True, cmap=plt.cm.Reds)
plt.show()
```

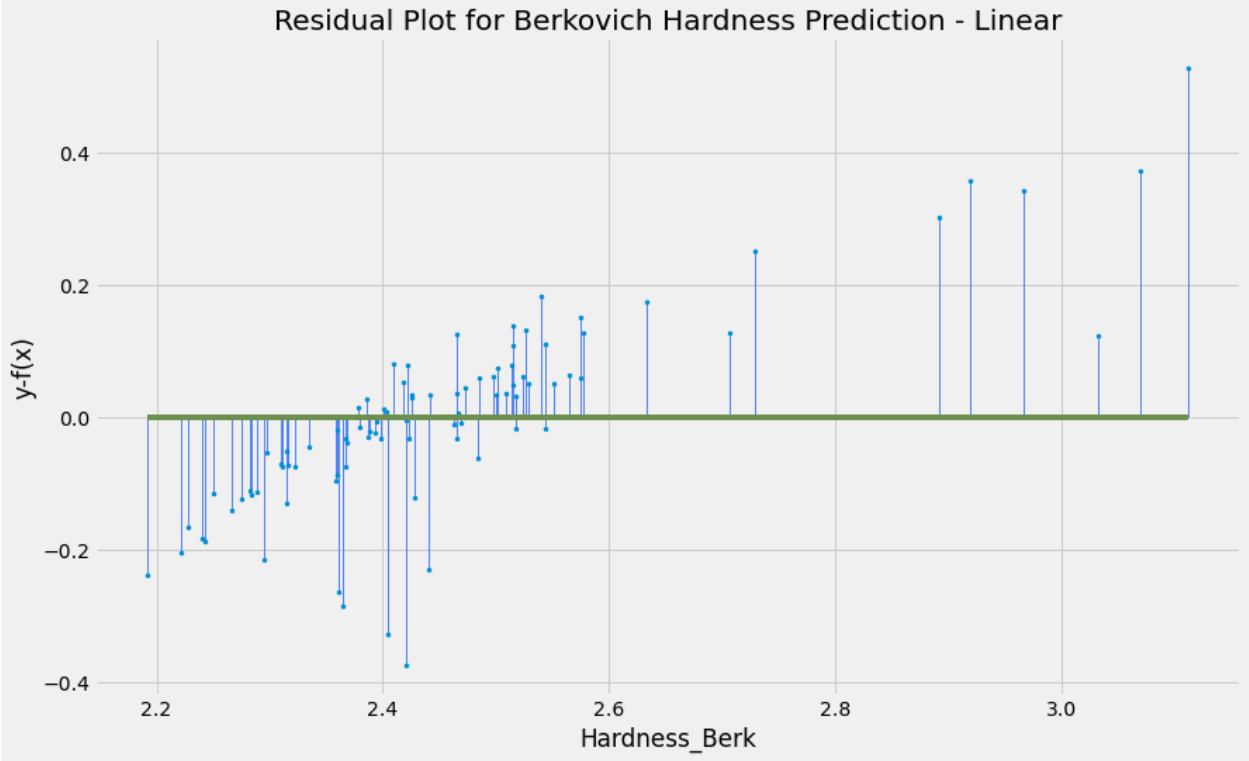

```
df= Given_data

model = linear_model.LinearRegression()

X1 = df[['Indentation Stress_Spherical (GPa)']]
Y1 = df['Hardness_Berk (GPa)']
```

```
X2 = df[['Modulus_Spherical (GPa)']]
Y2 = df['Modulus_Berk (GPa)']

model_hardness=model.fit(X1,Y1)

xfit1 = np.linspace(1.5,2.5,200)
Xfit1 = xfit1[:, np.newaxis] #this changes xfit from a horizontal
array to a vertical array!
yfit1 = model_hardness.predict(Xfit1)

Y_pred_hardness = model_hardness.predict(X1)

plt.figure(figsize = (5,4))
plt.scatter(X1,Y1,c='orange', label='Original Datapoints')
plt.plot(xfit1,yfit1, c='blue', label='Linear plot')
plt.ylabel('Indentation Stress_Spherical (GPa)')
plt.xlabel('Hardness_Berkovich (GPa)')
plt.legend()

r_2_1 = r2_score(Y1, Y_pred_hardness)
mae1 = mean_absolute_error(Y1, Y_pred_hardness)
mse1 = mean_squared_error(Y1, Y_pred_hardness)
rmse1 = np.sqrt(mse1) # or mse**(0.5)
print('MAE:',mae1,'MSE:',mse1, 'RMSE:', rmse1, 'R^2:',r_2_1)

plt.figure(figsize=(13,8))
plt.title('Parity Plot of Original Berkovich Hardness against Hardness
predicted by Model')
plt.scatter(Y1,Y_pred_hardness, s=2)
plt.xlabel('Original')
plt.ylabel('Model')
plt.show()

plt.figure(figsize=(13,8))
plt.title('Residual Plot for Berkovich Hardness Prediction - Linear')
plt.xlabel('Hardness_Berk')
plt.ylabel('y-f(x)')
plt.ylabel('y-f(x)')
markerline, stemline, baseline = plt.stem(Y1,
                                          Y1-Y_pred_hardness,
use_line_collection=True)
# plt.stem(Y1, Y1-Y_pred_hardness, use_line_collection=True)
plt.setp(stemline, linewidth = 1, color='#5284F2')
plt.setp(markerline, markersize = 2.5)
plt.show()
```

```
MAE: 0.1075721241281924 MSE: 0.02209616232154064 RMSE:
0.1486477794033286 R^2: 0.3189860990046359
```


Indentation Stress_Spherical (GPa)
3.0
2.8
2.6
2.4
2.2
1.6
1.8
2.0
2.2
2.4
Hardness_Berkovich (GPa)
Linear plot
Original Datapoints

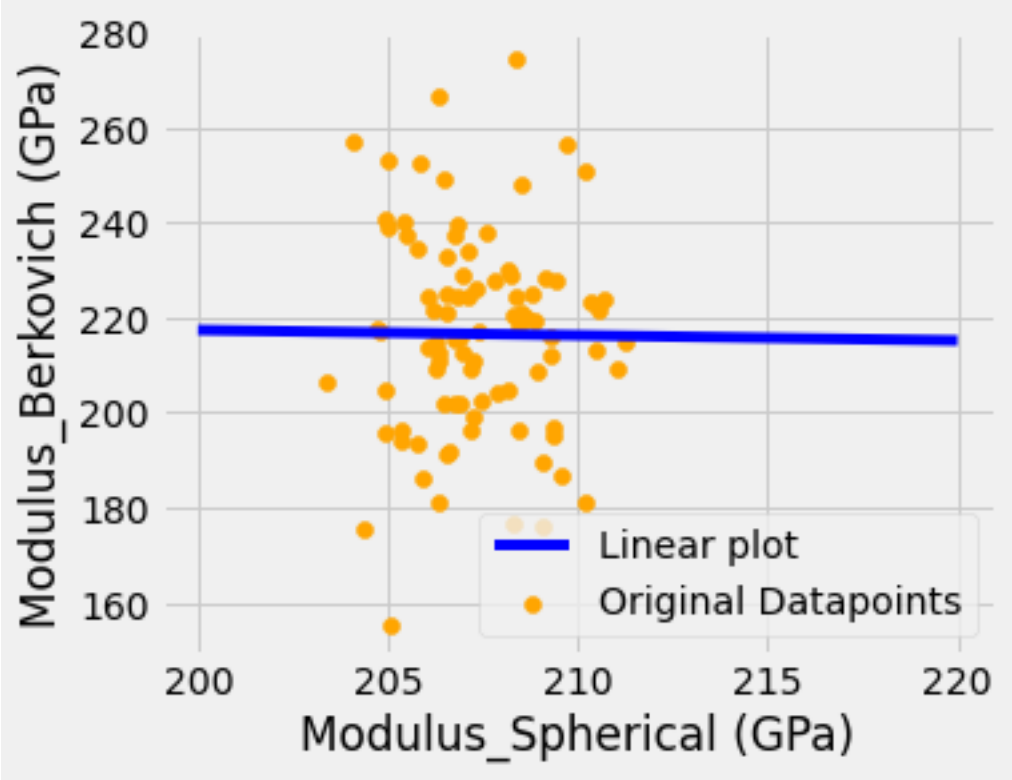

Parity Plot of Original Berkovich Hardness against Hardness predicted by Model
Model
2.9
2.8
2.7
2.6
2.5
2.4
2.2
2.4
2.6
2.8
3.0
Original

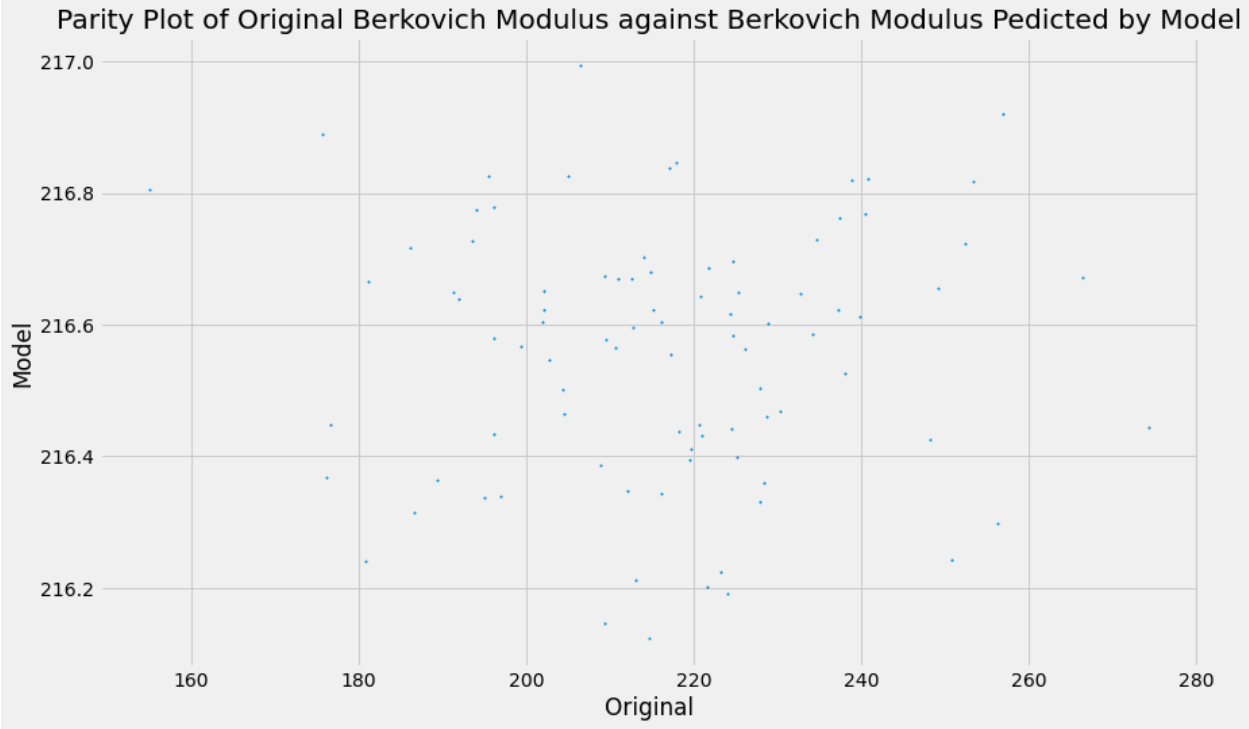


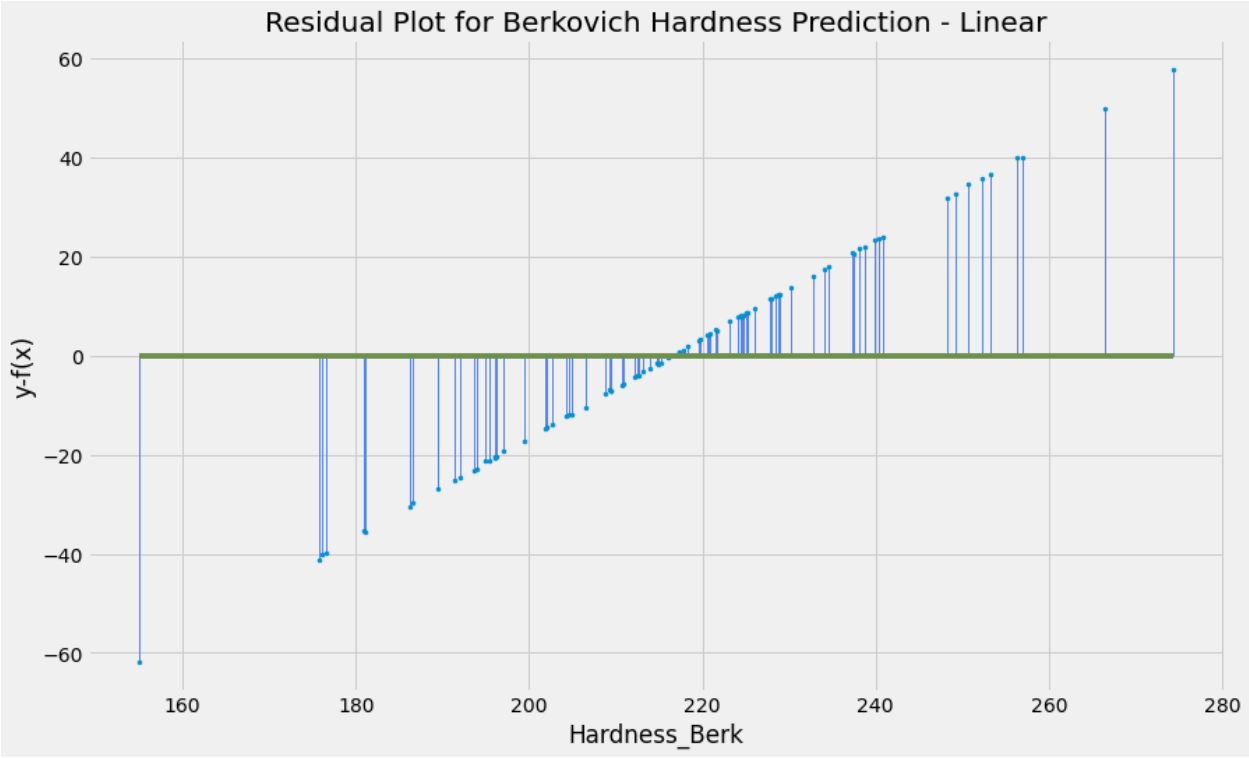

```
X2 = df[['Modulus_Spherical (GPa)']]
Y2 = df['Modulus_Berk (GPa)']
model_modulus=model.fit(X2,Y2)

xfit2 = np.linspace(200,220,200)
Xfit2 = xfit2[:, np.newaxis] #this changes xfit from a horizontal
array to a vertical array!
yfit2 = model_hardness.predict(Xfit2)

Y_pred_modulus = model_modulus.predict(X2)

plt.figure(figsize = (5,4))
plt.scatter(X2,Y2,c='orange', label='Original Datapoints')
plt.plot(xfit2,yfit2, c='blue', label='Linear plot')
plt.ylabel('Modulus_Berkovich (GPa)')
plt.xlabel('Modulus_Spherical (GPa)')
plt.legend()

r_2_2 = r2_score(Y2, Y_pred_modulus)
mae2= mean_absolute_error(Y2, Y_pred_modulus)
mse2= mean_squared_error(Y2, Y_pred_modulus)
rmse2= np.sqrt(mse2) # or mse**(0.5)
print('MAE:',mae2,'MSE:',mse2, 'RMSE:', rmse2, 'R^2:',r_2_2)


plt.figure(figsize=(13,8))
plt.title('Parity Plot of Original Berkovich Modulus against Berkovich
```

```
Modulus Pedicted by Model')
plt.scatter(Y2,Y_pred_modulus, s=2)
plt.xlabel('Original')
plt.ylabel('Model')
plt.show()

plt.figure(figsize=(13,8))
plt.title('Residual Plot for Berkovich Hardness Prediction - Linear')
plt.xlabel('Hardness_Berk')
plt.ylabel('y-f(x)')
plt.ylabel('y-f(x)')
markerline, stemline, baseline = plt.stem(Y2,
                                          Y2-Y_pred_modulus,
use_line_collection=True)
plt.setp(stemline, linewidth = 1, color='#5284F2')
plt.setp(markerline, markersize = 2.5)
plt.show()
```

```
MAE: 16.758597428010237 MSE: 466.4778228559398 RMSE: 21.59809766752479
R^2: 7.955499296519175e-05
```



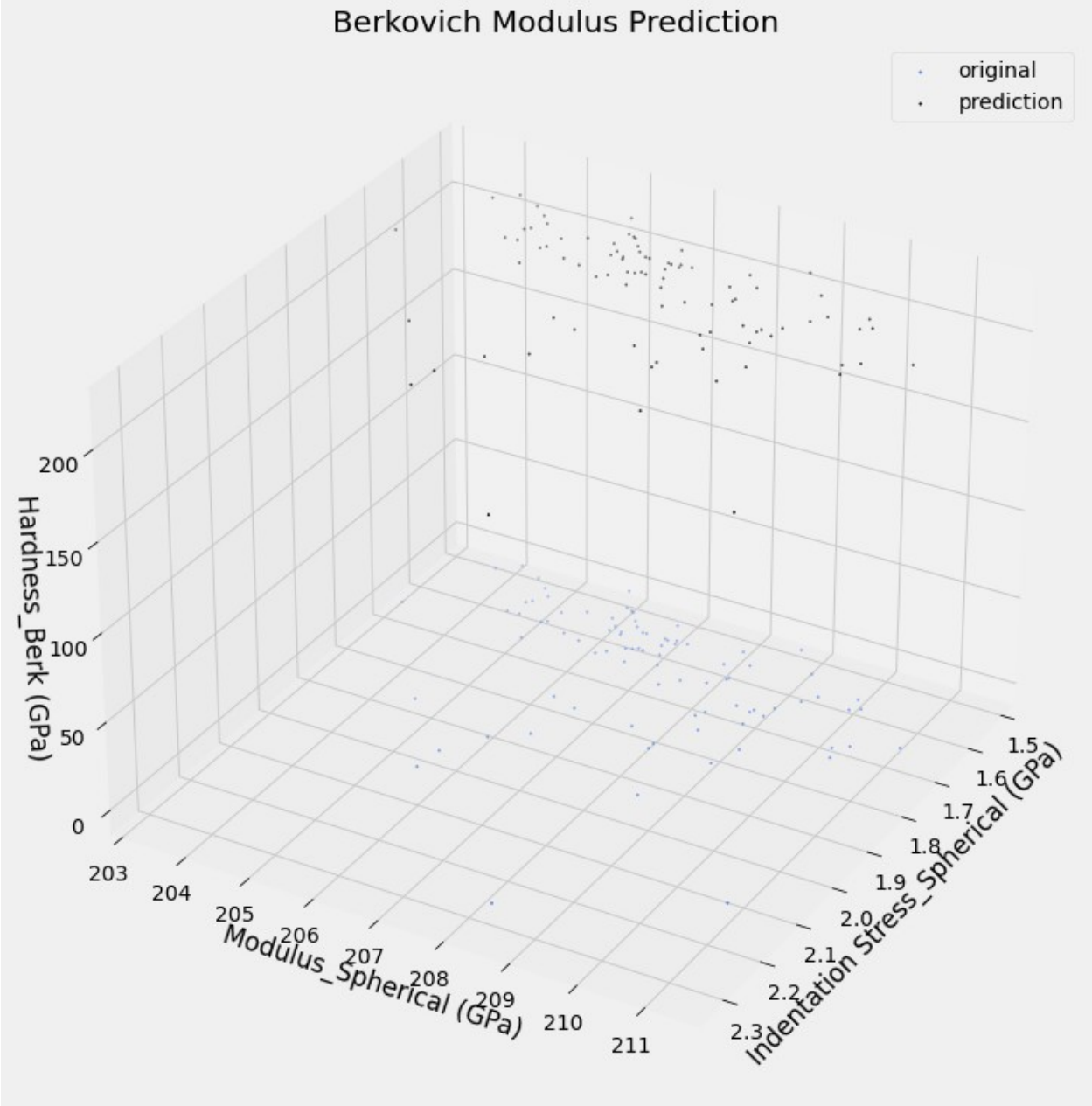





```
Shperical_X  = df[['Indentation Stress_Spherical
(GPa)','Modulus_Spherical (GPa)']]
#Shperical_X  = (Shperical_X - np.mean(Shperical_X, axis = 0))/
np.std(Shperical_X, axis = 0)
```

```
Berk_Hardness_Y2 = df['Hardness_Berk (GPa)']
#Berk_Hardness_Y2  = (Berk_Hardness_Y2  - np.mean(Berk_Hardness_Y2 ,
axis = 0))/ np.std(Berk_Hardness_Y2, axis = 0)

Berk_Modulus_Y2 = df['Modulus_Berk (GPa)']
#Berk_Modulus_Y2  = (Berk_Modulus_Y2  - np.mean(Berk_Modulus_Y2 , axis
= 0))/ np.std(Berk_Modulus_Y2, axis = 0)

# Tranining the model (finding the fit)
model2_mod = model.fit(Shperical_X,Berk_Modulus_Y2)

#Step 5
plt.figure(figsize=(15,12))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(Shperical_X['Indentation Stress_Spherical
(GPa)'],Shperical_X['Modulus_Spherical (GPa)'],
                   Berk_Hardness_Y2, s=2,
                   c='#5284F2', label='original')
plot2=ax.scatter3D(Shperical_X['Indentation Stress_Spherical
(GPa)'],Shperical_X['Modulus_Spherical (GPa)'],
                   model2_mod.predict(Shperical_X), c='black', s=2,
                   label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(30,30)
ax.set_xlabel('Indentation Stress_Spherical (GPa)')
ax.set_ylabel('Modulus_Spherical (GPa)')
ax.set_zlabel('Hardness_Berk (GPa)')
# ax.set_zlim(0,15)
ax.legend()
ax.set_title('Berkovich Modulus Prediction')
```

```
Text(0.5, 0.92, 'Berkovich Modulus Prediction')
```

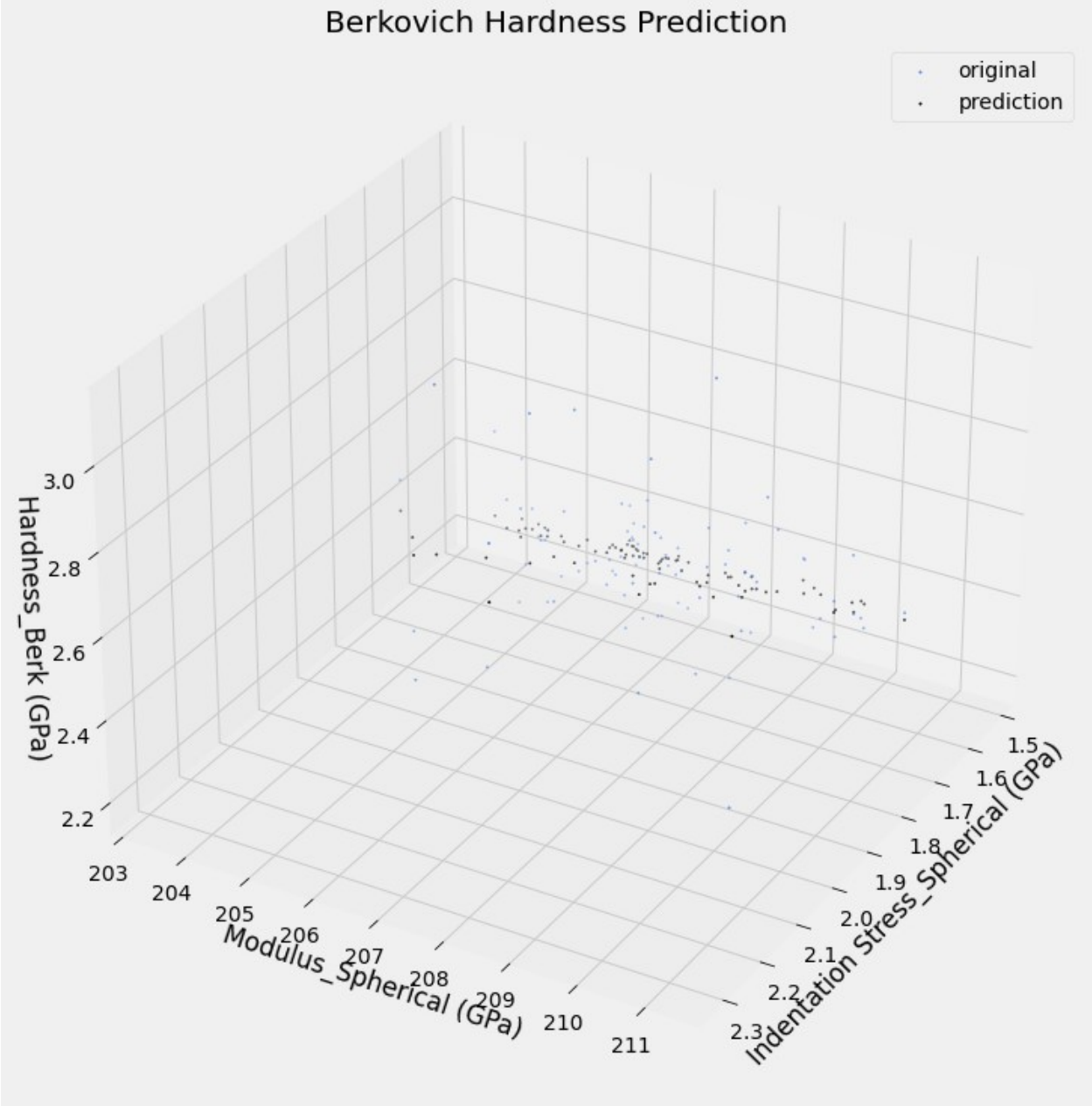



```
mae3 = mean_absolute_error(Berk_Modulus_Y2 ,
model2_mod.predict(Shperical_X))
mse3 = mean_squared_error(Berk_Modulus_Y2 ,
model2_mod.predict(Shperical_X))
rmse3 = np.sqrt(mse3) # or mse**(0.5)
r2_3 = r2_score(Berk_Modulus_Y2, model2_mod.predict(Shperical_X))

print('MAE:',mae3,'MSE:',mse3, 'RMSE:', rmse3, 'R^2:',r2_3)
```

```
MAE: 16.76740026430189 MSE: 466.1193852455087 RMSE: 21.58979817519165
R^2: 0.0008478853987609059
```

```
model3_hard = model.fit(Shperical_X,Berk_Hardness_Y2)

#Step 5
plt.figure(figsize=(15,12))
ax = plt.axes(projection = '3d')
ax = plt.axes(projection='3d')
plot1=ax.scatter3D(Shperical_X['Indentation Stress_Spherical
(GPa)'],Shperical_X['Modulus_Spherical (GPa)'],
                   Berk_Hardness_Y2, s=2,
                   c='#5284F2', label='original')
plot2=ax.scatter3D(Shperical_X['Indentation Stress_Spherical
(GPa)'],Shperical_X['Modulus_Spherical (GPa)'],
                   model3_hard.predict(Shperical_X), c='black', s=2,
                   label='prediction')
#as always, play around with these angles to explore the fitted
plane/surface
ax.view_init(30,30)
ax.set_xlabel('Indentation Stress_Spherical (GPa)')
ax.set_ylabel('Modulus_Spherical (GPa)')
ax.set_zlabel('Hardness_Berk (GPa)')
# ax.set_zlim(0,15)
ax.legend()
ax.set_title('Berkovich Hardness Prediction')
```

```
Text(0.5, 0.92, 'Berkovich Hardness Prediction')
```




```
mae4 = mean_absolute_error(Berk_Hardness_Y2 ,
model3_hard.predict(Shperical_X))
mse4= mean_squared_error(Berk_Hardness_Y2 ,
model3_hard.predict(Shperical_X))
rmse4 = np.sqrt(mse4) # or mse**(0.5)
r2_4 = r2_score(Berk_Hardness_Y2, model3_hard.predict(Shperical_X))
print('MAE:',mae4,'MSE:',mse4, 'RMSE:', rmse4, 'R^2:',r2_4)
```

```
MAE: 0.10578147856913377 MSE: 0.02175278967271485 RMSE:
0.14748826961055192 R^2: 0.32956900221059215
```

Comparison of error function for each algorithm

- Following are the comparison of error functions for each algorithm
  - Linear Regression of Spherical Indentation Stress and Berkovich Hardness (Feature: Spherical Indentation Stress, Target: Berkovich Hardness; Results: MAE: 0.1075, MSE: 0.02209, RMSE: 0.8187, R^2: 0.31898)
  - Linear Regression of Spherical Modulus and Berkovich Modulus (Feature: Spherical Modulus, Target: Berkovich Modulus; Results: MAE: 16.75, MSE: 466.47, RMSE: 0.818, R^2: 7.95e-05)
  - Multi-linear Regression of Spherical Indentation Stress and Berkovich Hardness (Feature: Spherical Indentation Stress and Spherical Modulus, Target: Berkovich Modulus; Results: MAE: 16.76, MSE: 466.11, RMSE: 0.8188, R^2: 0.00084)
  - Multi-linear Regression of Spherical Modulus and Berkovich Modulus(Feature: Spherical Indentation Stress and Spherical Modulus, Target: Berkovich Hardness; Results: MAE: 0.1058, MSE: 0.0217, RMSE: 0.81879 R^2: 0.33)
- Both linear and multilinear regression fails to predict the berkovich modulus in as built condition
- Due to higher fidelity and better accuracy, multilinear regression in predicting Berkovich hardness was chose for further analysis to predict berkovich hardness of Tantalum in as-built condition. However, the dataset limited, with further inclusion of experimental data would result in better result

Calculation of Nano hardness in as built condition

```
df_As_Built = df[['Indentation Stress_As-Built (GPa)','Modulus_As-built_Spherical (GPa)']]
df_As_Built = df_As_Built.dropna()
model3_hard.predict(df_As_Built)

df_As_Built['Berkovich Hardness_As-Built (GPa)'] = model3_hard.predict(df_As_Built).tolist()

df_As_Built
```

```
    Indentation Stress_As-Built (GPa)  Modulus_As-built_Spherical (GPa)  \
0                            1.190366                        207.075287
1                            1.220127                        203.976579
2                            1.221094                        203.399301
3                            1.195815                        205.444007
4                            1.320454                        204.712112
5                            1.348287                        203.725349
6                            1.248875
```

```
204.658933
7                                   1.260260
205.544539
8                                   1.190645
204.689549
9                                   1.283854
206.676401
10                                  1.281056
204.049401
11                                  1.129765
205.128556
12                                  1.271668
206.133489
13                                  1.291872
204.853568
14                                  1.263403
207.225684
15                                  1.289000
204.380534
16                                  1.325083
204.517945
17                                  1.188682
208.001050
18                                  1.232777
203.975200
19                                  1.301200
206.712552
20                                  1.180779
204.787980
21                                  1.283752
205.911357
22                                  1.341529
206.230199
23                                  1.356726
205.895514
24                                  1.229200
206.925688
25                                  1.336149
205.834304
26                                  1.249445
206.172599
27                                  1.291664
206.933508
28                                  1.320588
205.375965
29                                  1.273266
205.444091
30                                  1.260312
205.683977
```

```
31                                1.261375
206.291730

    Berkovich Hardness_As-Built (GPa)
0                                 2.097346
1                                 2.084074
2                                 2.078392
3                                 2.083196
4                                 2.162508
5                                 2.171150
6                                 2.111744
7                                 2.129481
8                                 2.071260
9                                 2.158490
10                                2.127589
11                                2.033418
12                                2.143966
13                                2.144030
14                                2.150204
15                                2.136806
16                                2.163614
17                                2.106365
18                                2.092926
19                                2.171048
20                                2.065428
21                                2.149990
22                                2.194006
23                                2.200973
24                                2.122922
25                                2.185874
26                                2.128818
27                                2.166798
28                                2.169915
29                                2.137492
30                                2.131053
31                                2.138494
```

```
df_As_Built['Berkovich Hardness_As-Built (GPa)'].mean()
Y1 = df['Hardness_Berk (GPa)']
print('Average Berkovich Hardness in As_Built Condition
(GPa):',df_As_Built['Berkovich Hardness_As-Built (GPa)'].mean())
print('Average Berkovich Hardness in Shock Loaded Condition (GPa):',
df['Hardness_Berk (GPa)'].mean())
```

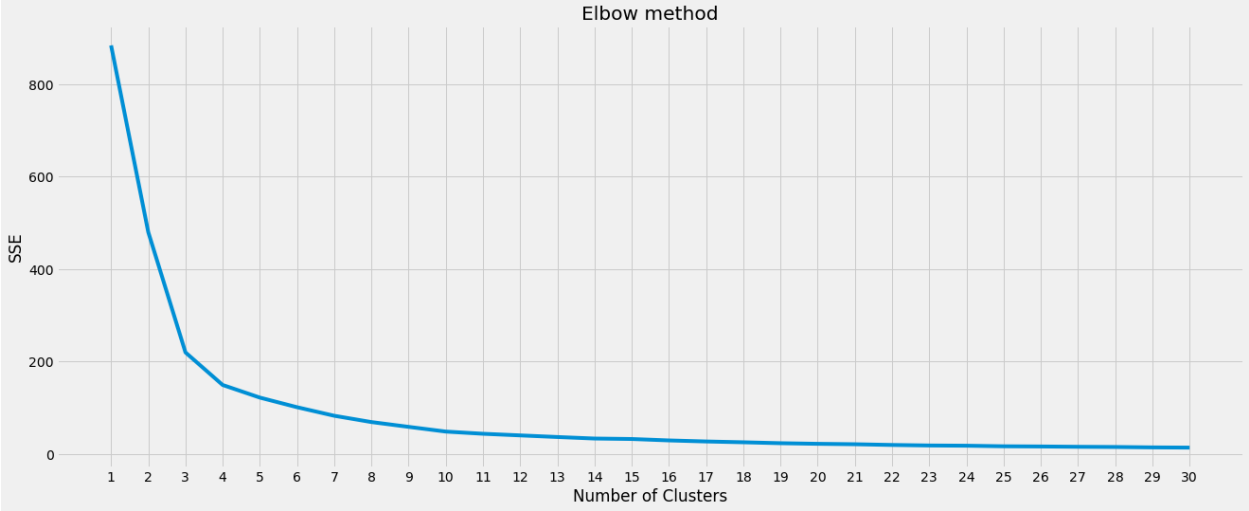
```
Average Berkovich Hardness in As_Built Condition (GPa):
2.131542738286477
Average Berkovich Hardness in Shock Loaded Condition (GPa):
2.460082047449437
```

- Shock loading increases the hardness after electron beam printing of Tantalum

- Average berkovich hardness in as-built condition is 2.13 GPa and Shock loaded condition is 2.46 GPa
- This means more plasticity is induced in due to shock loading
- The nanoindenation of conventional (not additive manufactured) tanatlum is reported to be ranged between 2-4.5 GPa, depending on orientation [X]. Thus, our result is good enough, also falls within a reasonable range of the value of the experimental nanohardness in this study.

[Ref- X] T.P. Remington, C.J. Ruestes, E.M. Bringa, B.A. Remington, C.H. Lu, B. Kad, M.A. Meyers, Plastic deformation in nanoindentation of tantalum: A new mechanism for prismatic loop formation, Acta Materialia,Volume 78,2014,Pages 378-393,ISSN 1359-6454,https://doi.org/10.1016/j.actamat.2014.06.058.

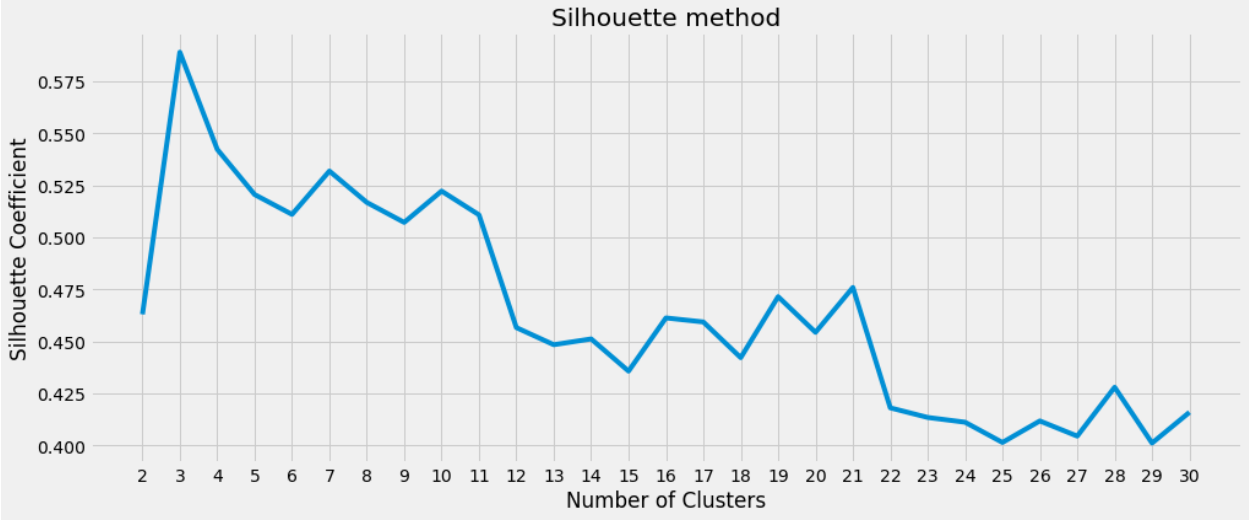
```
data = pd.read_csv("./hardness_indentation.csv")
subset = data[['Modulus (GPa)', 'Hardness (Gpa)']]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(subset) #this will scale the
data for each feature that the means are 0, and variances are 1.

# A list holds the SSE values for each k
sse = []
for k in range(1, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    sse.append(clusterer.inertia_)

plt.figure(figsize=(20,8))
plt.style.use("fivethirtyeight")
plt.plot(range(1, 31), sse)
plt.xticks(range(1, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Elbow method")
plt.show()
```




```
## Find the elbow point
kl = KneeLocator(
    range(1, 31), sse, curve="convex", direction="decreasing")
print("Optimal number of clusters based on elbow method: ", kl.elbow)
```

```
Optimal number of clusters based on elbow method:  4
```

```
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []
for k in range(2, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    score = silhouette_score(scaled_features, clusterer.labels_)
    silhouette_coefficients.append(score)

plt.figure(figsize=(15,6))
plt.style.use("fivethirtyeight")
plt.plot(range(2, 31), silhouette_coefficients)
plt.xticks(range(2, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.title("Silhouette method")
plt.show()
```

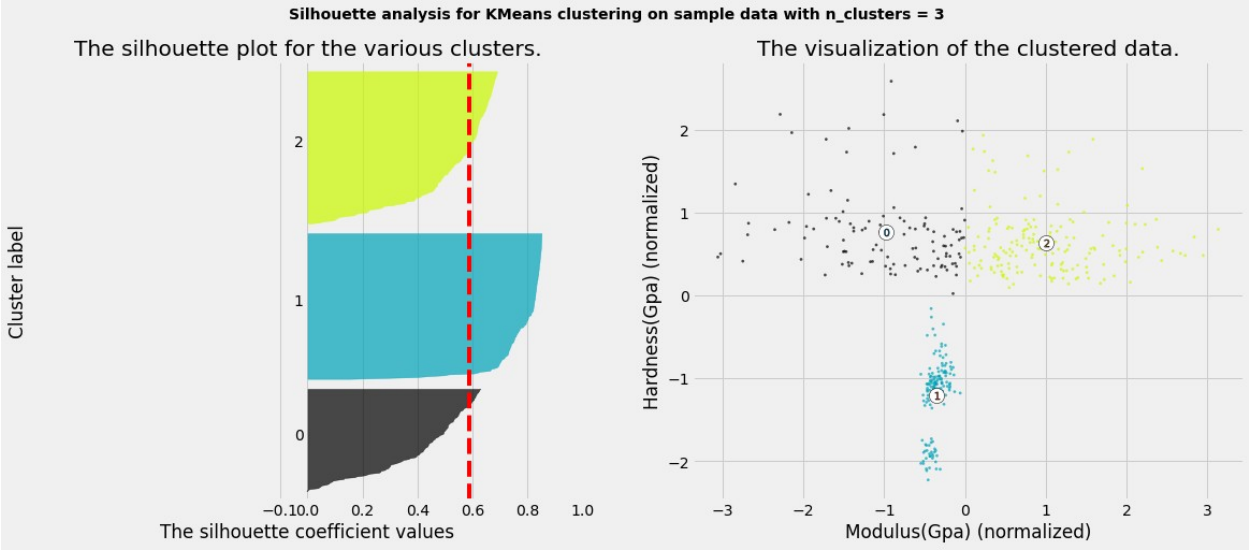



```
# let's take a look at what silhouette analysis looks like:
X1 = scaled_features
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters , random_state=55)
kmeans.fit(scaled_features)
cluster_labels = kmeans.labels_
score = silhouette_score(scaled_features, cluster_labels)

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_size_inches(18, 7)

# The 1st subplot is the silhouette plot
# The silhouette coefficient can range from -1, 1 but in this example
all
# lie within [-0.1, 1]
ax1.set_xlim([-1, 1])
# The (n_clusters+1)*10 is for inserting blank space between
silhouette
# plots of individual clusters, to demarcate them clearly.
ax1.set_ylim([0, len(scaled_features) + (n_clusters + 1) * 10])


# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the
formed
# clusters
silhouette_avg = silhouette_score(scaled_features, cluster_labels)
print("For n_clusters =", n_clusters,
    "The average silhouette_score is :", silhouette_avg)

# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(scaled_features,
cluster_labels)
```

```
y_lower = 10

for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the
middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1,0, 0.2, 0.4, 0.6, 0.8, 1])

# 2nd Plot showing the actual clusters formed
colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
ax2.scatter(X1[:, 0], X1[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

# Labeling the clusters
centers = kmeans.cluster_centers_
# Draw white circles at cluster centers
ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
            c="white", alpha=1, s=200, edgecolor='k')

for i, c in enumerate(centers):
    ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                      s=50, edgecolor='k')
```

```
ax2.set_title("The visualization of the clustered data.")
ax2.set_xlabel("Modulus(Gpa) (normalized)")
ax2.set_ylabel("Hardness(Gpa) (normalized)")

plt.suptitle(("Silhouette analysis for KMeans clustering on sample
data "
              "with n_clusters = %d" % n_clusters),
              fontsize=14, fontweight='bold')

plt.show()
```

```
For n_clusters = 3 The average silhouette_score is : 0.589021191507476
```

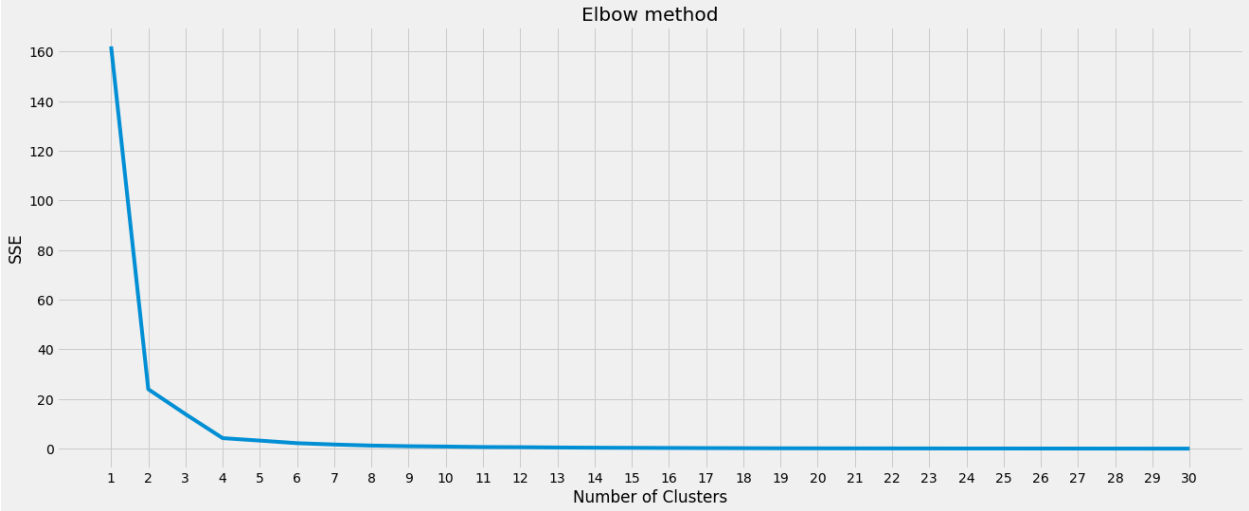



```
label_df =
pd.DataFrame(cluster_labels,columns=['kmeans_cluster_label'])
result = pd.concat([data,label_df],axis=1)
result.to_csv('modulus_hardness_cluster.csv',index=False)
```

The above block saves the labels as a `.csv` file and we see that the last column shows to which cluster they have been put in.
Now we will be working with `GB1 Data on As_Build Condition based on Hardness Values`.

```
data = pd.read_csv("./Hardness_Modulus_Asbuilt_GB1.csv")
subset = data[['Hardness (Gpa)']]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(subset) #this will scale the
data for each feature that the means are 0, and variances are 1.

sse = []
for k in range(1, 31):
```

```
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    sse.append(clusterer.inertia_)

plt.figure(figsize=(20,8))
plt.style.use("fivethirtyeight")
plt.plot(range(1, 31), sse)
plt.xticks(range(1, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Elbow method")
plt.show()
```

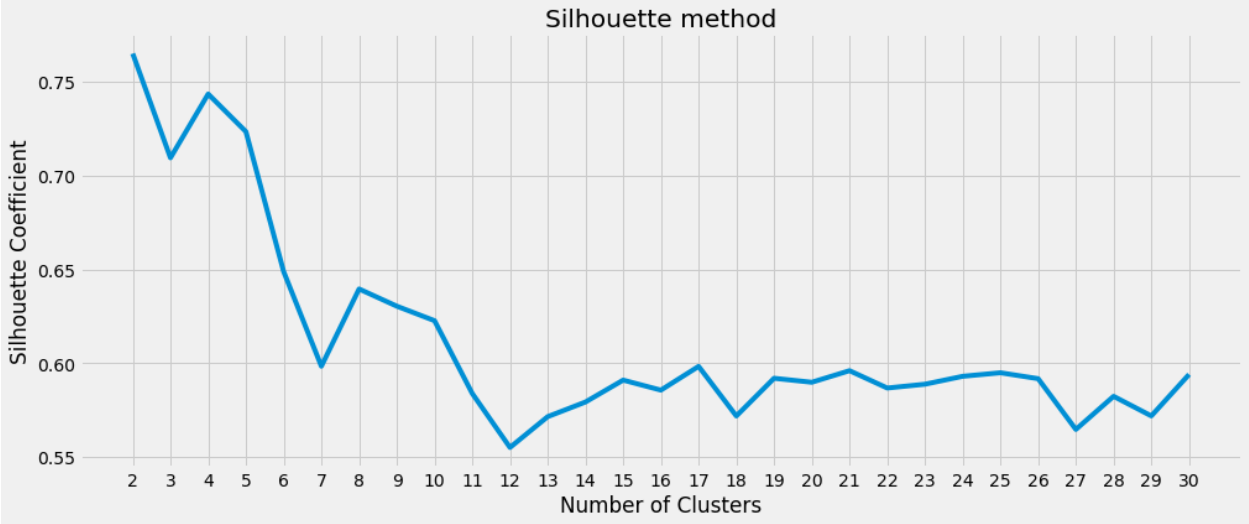



```
## Find the elbow point
kl = KneeLocator(
    range(1, 31), sse, curve="convex", direction="decreasing")
print("Optimal number of clusters based on elbow method: ", kl.elbow)
```

```
Optimal number of clusters based on elbow method:  4
```

```
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []
for k in range(2, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    score = silhouette_score(scaled_features, clusterer.labels_)
    silhouette_coefficients.append(score)

plt.figure(figsize=(15,6))
plt.style.use("fivethirtyeight")
plt.plot(range(2, 31), silhouette_coefficients)
plt.xticks(range(2, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
```

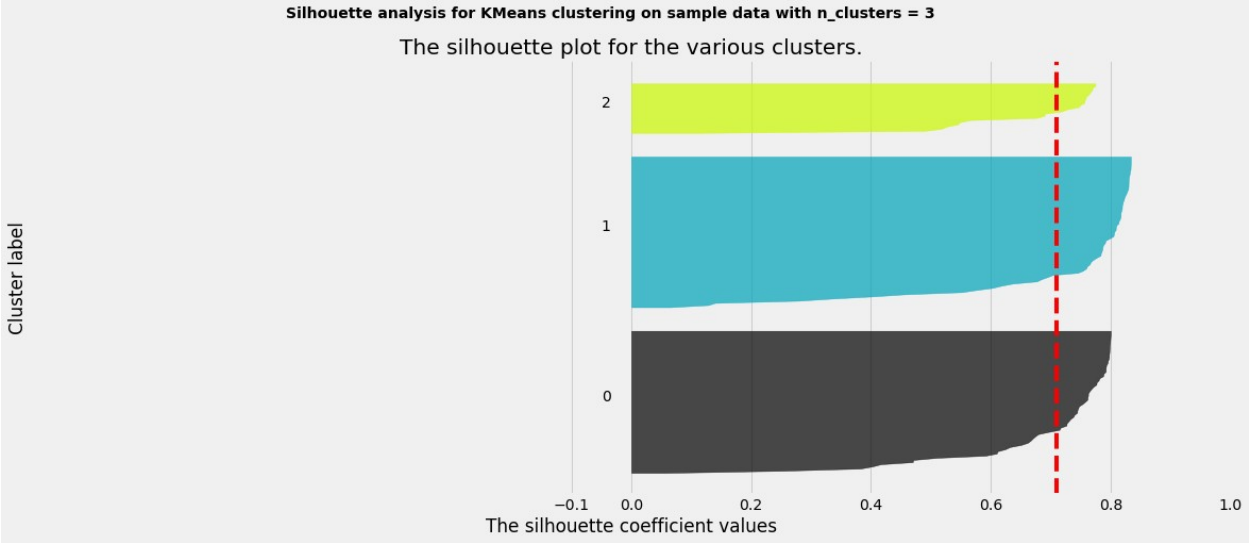
```
plt.title("Silhouette method")
plt.show()
```




```
# let's take a look at what silhouette analysis looks like:
X1 = scaled_features
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters , random_state=55)
kmeans.fit(scaled_features)
cluster_labels = kmeans.labels_
score = silhouette_score(scaled_features, cluster_labels)

fig, ax1 = plt.subplots()
fig.set_size_inches(18, 7)

# The 1st subplot is the silhouette plot
# The silhouette coefficient can range from -1, 1 but in this example
all
# lie within [-0.1, 1]
ax1.set_xlim([-1, 1])
# The (n_clusters+1)*10 is for inserting blank space between
silhouette
# plots of individual clusters, to demarcate them clearly.
ax1.set_ylim([0, len(scaled_features) + (n_clusters + 1) * 10])


# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the
formed
# clusters
silhouette_avg = silhouette_score(scaled_features, cluster_labels)
print("For n_clusters =", n_clusters,
    "The average silhouette_score is :", silhouette_avg)
```

```
# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(scaled_features,
cluster_labels)

y_lower = 10

for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the
middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1,0, 0.2, 0.4, 0.6, 0.8, 1])

plt.suptitle(("Silhouette analysis for KMeans clustering on sample
data "
              "with n_clusters = %d" % n_clusters),fontsize=14,
fontweight='bold')

plt.show()
```

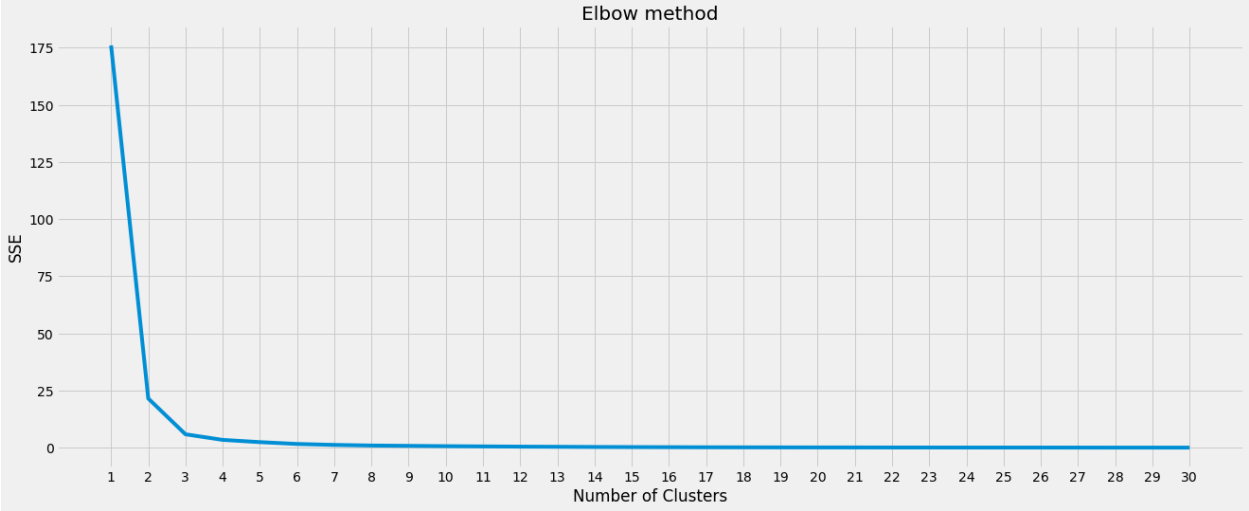
```
For n_clusters = 3 The average silhouette_score is :
0.7094566132224285
```




```
label_df =
pd.DataFrame(cluster_labels,columns=['kmeans_cluster_label'])
result = pd.concat([data,label_df],axis=1)
result.to_csv('kmeans_result_Asbuilt_GB1.csv',index=False)
```

The above block saves the labels as a `.csv` file and we see that the last column shows to which cluster they have been put in.
Now we will be working with `GB2 Data on As_Build Condition based on Hardness Values`.

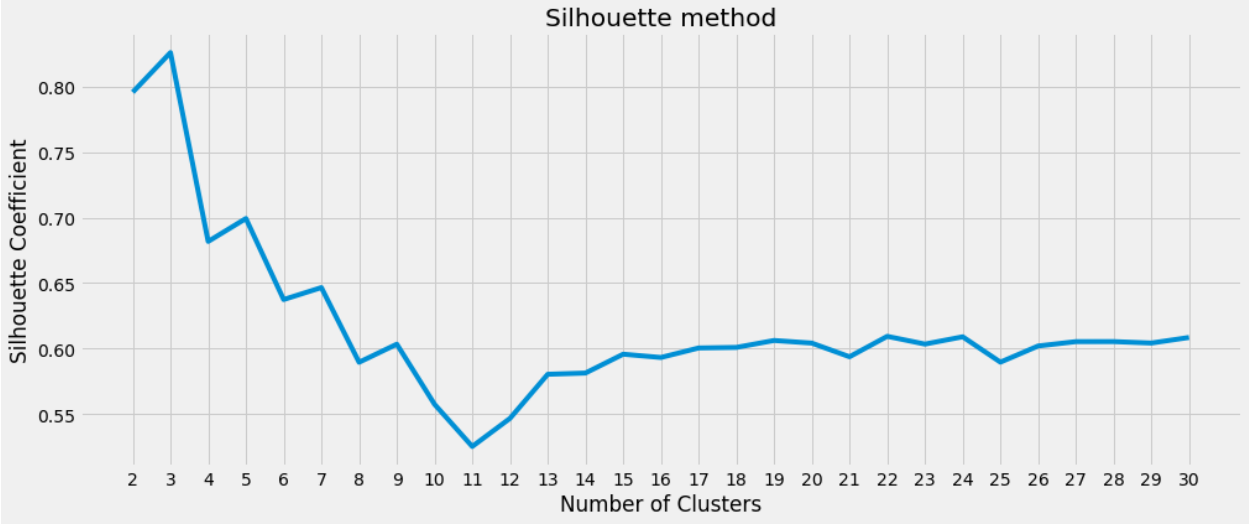
```
data = pd.read_csv("./Hardness_Modulus_Asbuilt_GB2.csv")
subset = data[['Hardness (Gpa)']]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(subset) #this will scale the
data for each feature that the means are 0, and variances are 1.

# A list holds the SSE values for each k
sse = []
for k in range(1, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    sse.append(clusterer.inertia_)

plt.figure(figsize=(20,8))
plt.style.use("fivethirtyeight")
plt.plot(range(1, 31), sse)
plt.xticks(range(1, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Elbow method")
plt.show()
```




```
## Find the elbow point
kl = KneeLocator(
    range(1, 31), sse, curve="convex", direction="decreasing")
print("Optimal number of clusters based on elbow method: ", kl.elbow)
```

```
Optimal number of clusters based on elbow method:  3
```

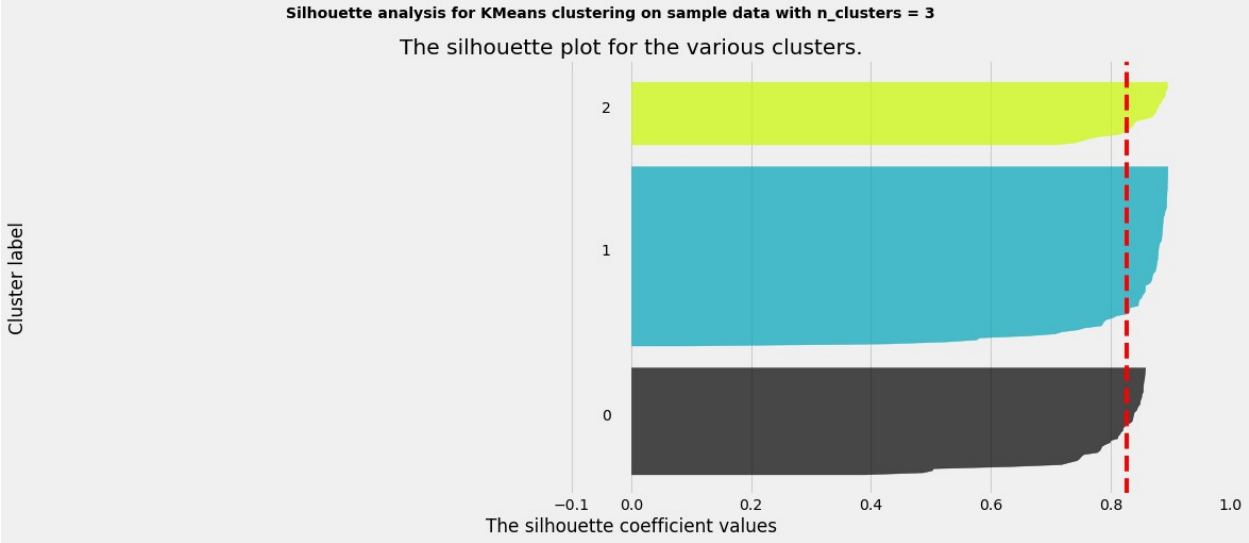
```
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []
for k in range(2, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    score = silhouette_score(scaled_features, clusterer.labels_)
    silhouette_coefficients.append(score)

plt.figure(figsize=(15,6))
plt.style.use("fivethirtyeight")
plt.plot(range(2, 31), silhouette_coefficients)
plt.xticks(range(2, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.title("Silhouette method")
plt.show()
```




```
# let's take a look at what silhouette analysis looks like:
X1 = scaled_features
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters , random_state=55)
kmeans.fit(scaled_features)
cluster_labels = kmeans.labels_
score = silhouette_score(scaled_features, cluster_labels)

fig, ax1 = plt.subplots()
fig.set_size_inches(18, 7)

# The 1st subplot is the silhouette plot
# The silhouette coefficient can range from -1, 1 but in this example
all
# lie within [-0.1, 1]
ax1.set_xlim([-1, 1])
# The (n_clusters+1)*10 is for inserting blank space between
silhouette
# plots of individual clusters, to demarcate them clearly.
ax1.set_ylim([0, len(scaled_features) + (n_clusters + 1) * 10])


# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the
formed
# clusters
silhouette_avg = silhouette_score(scaled_features, cluster_labels)
print("For n_clusters =", n_clusters,
    "The average silhouette_score is :", silhouette_avg)

# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(scaled_features,
cluster_labels)
```

```
y_lower = 10

for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the
middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1,0, 0.2, 0.4, 0.6, 0.8, 1])

plt.suptitle(("Silhouette analysis for KMeans clustering on sample
data "
              "with n_clusters = %d" % n_clusters),fontsize=14,
fontweight='bold')

plt.show()
```

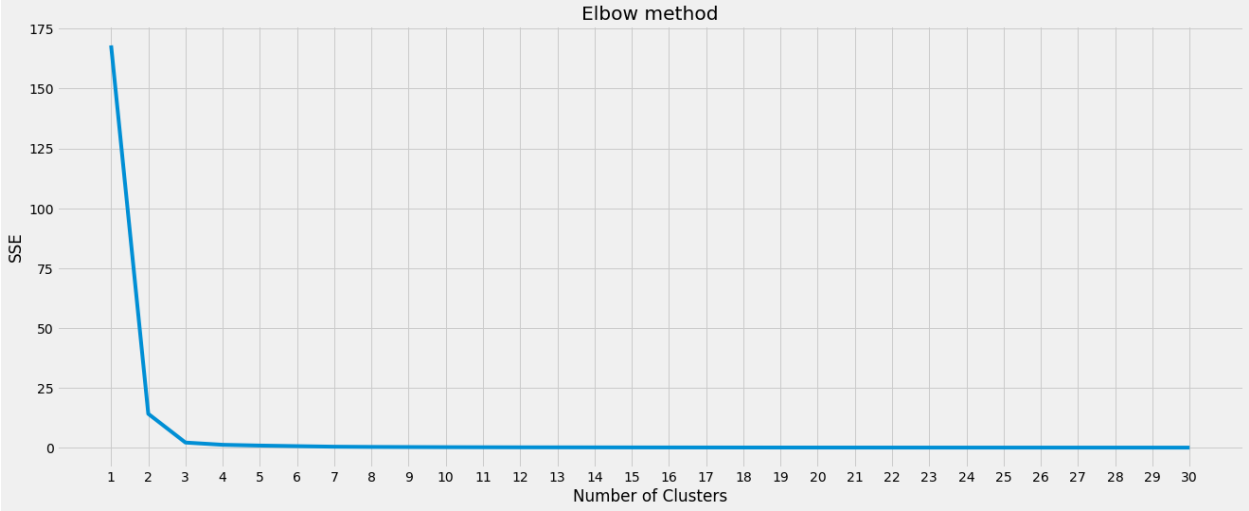
```
For n_clusters = 3 The average silhouette_score is :
0.8258856306940445
```




```
label_df =
pd.DataFrame(cluster_labels,columns=['kmeans_cluster_label'])
result = pd.concat([data,label_df],axis=1)
result.to_csv('kmeans_result_Asbuilt_GB2.csv',index=False)
```

The above block saves the labels as a `.csv` file and we see that the last column shows to which cluster they have been put in.
Now we will be working with `GB3 Data on As_Build Condition based on Hardness Values`.

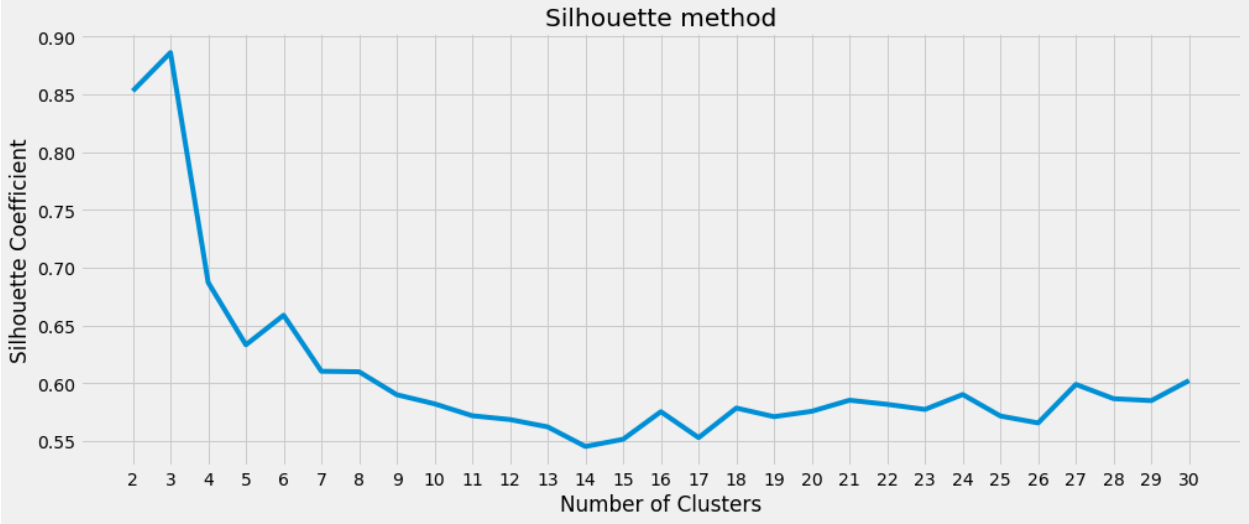
```
data = pd.read_csv("./Hardness_Modulus_Asbuilt_GB3.csv")
subset = data[['Hardness (Gpa)']]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(subset) #this will scale the
data for each feature that the means are 0, and variances are 1.

# A list holds the SSE values for each k
sse = []
for k in range(1, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    sse.append(clusterer.inertia_)

plt.figure(figsize=(20,8))
plt.style.use("fivethirtyeight")
plt.plot(range(1, 31), sse)
plt.xticks(range(1, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Elbow method")
plt.show()
```




```
## Find the elbow point
kl = KneeLocator(
    range(1, 31), sse, curve="convex", direction="decreasing")
print("Optimal number of clusters based on elbow method: ", kl.elbow)
```

```
Optimal number of clusters based on elbow method:  3
```

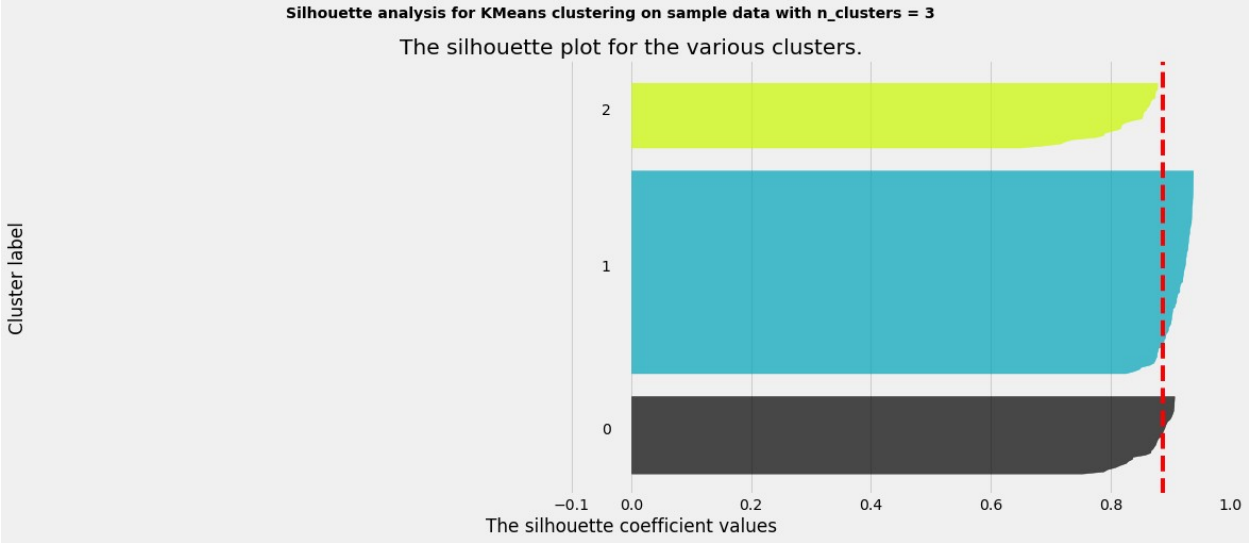
```
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []
for k in range(2, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    score = silhouette_score(scaled_features, clusterer.labels_)
    silhouette_coefficients.append(score)

plt.figure(figsize=(15,6))
plt.style.use("fivethirtyeight")
plt.plot(range(2, 31), silhouette_coefficients)
plt.xticks(range(2, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.title("Silhouette method")
plt.show()
```




```
# let's take a look at what silhouette analysis looks like:
X1 = scaled_features
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters , random_state=55)
kmeans.fit(scaled_features)
cluster_labels = kmeans.labels_
score = silhouette_score(scaled_features, cluster_labels)

fig, ax1 = plt.subplots()
fig.set_size_inches(18, 7)

# The 1st subplot is the silhouette plot
# The silhouette coefficient can range from -1, 1 but in this example
all
# lie within [-0.1, 1]
ax1.set_xlim([-1, 1])
# The (n_clusters+1)*10 is for inserting blank space between
silhouette
# plots of individual clusters, to demarcate them clearly.
ax1.set_ylim([0, len(scaled_features) + (n_clusters + 1) * 10])


# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the
formed
# clusters
silhouette_avg = silhouette_score(scaled_features, cluster_labels)
print("For n_clusters =", n_clusters,
    "The average silhouette_score is :", silhouette_avg)

# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(scaled_features,
cluster_labels)
```

```
y_lower = 10

for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the
middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1,0, 0.2, 0.4, 0.6, 0.8, 1])

plt.suptitle(("Silhouette analysis for KMeans clustering on sample
data "
              "with n_clusters = %d" % n_clusters),fontsize=14,
fontweight='bold')

plt.show()
```

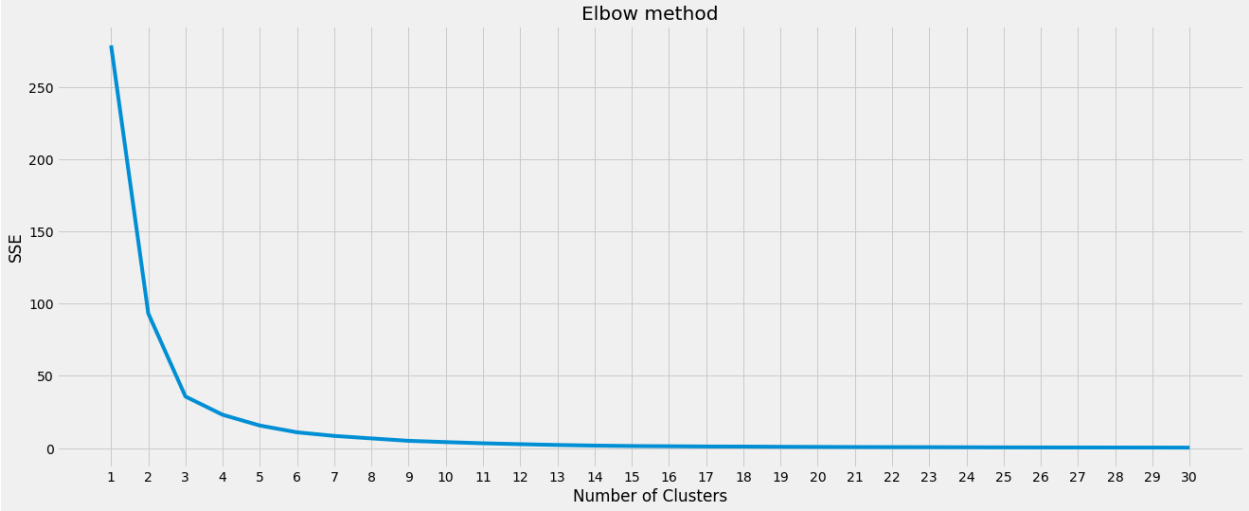
```
For n_clusters = 3 The average silhouette_score is :
0.8862432670539714
```




```
label_df =
pd.DataFrame(cluster_labels,columns=['kmeans_cluster_label'])
result = pd.concat([data,label_df],axis=1)
result.to_csv('kmeans_result_Asbuilt_GB3.csv',index=False)
```

The above block saves the labels as a `.csv` file and we see that the last column shows to which cluster they have been put in.
Now we will be working with `All the GB data for Berkovich Nano-indentation testing`.

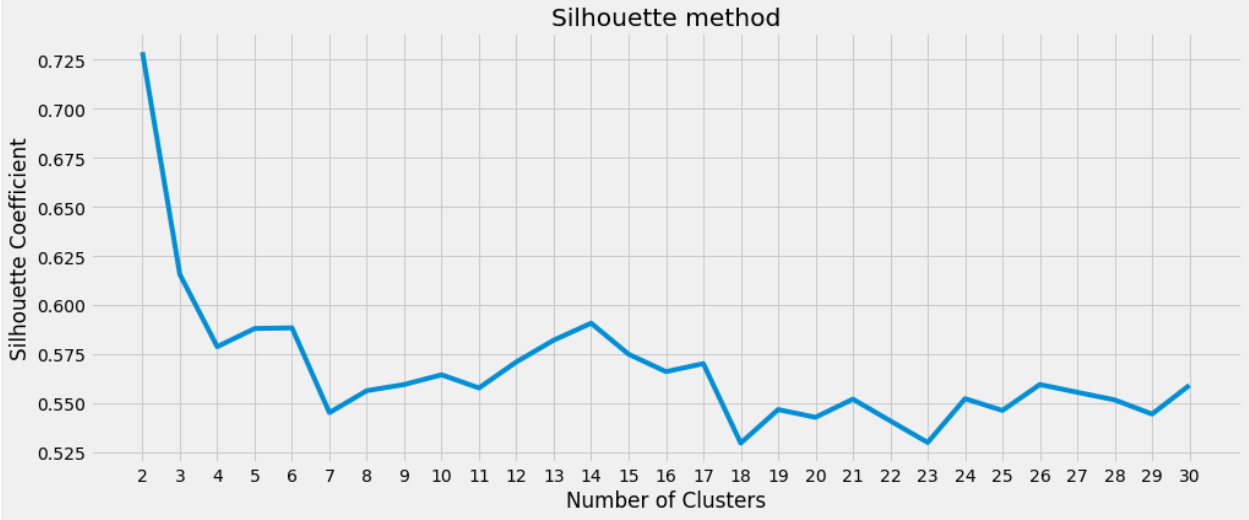
```
data = pd.read_csv("./Hardness_Modulus_Berk_all_GB.csv")
subset = data[['Hardness (Gpa)']]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(subset) #this will scale the
data for each feature that the means are 0, and variances are 1.

# A list holds the SSE values for each k
sse = []
for k in range(1, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    sse.append(clusterer.inertia_)

plt.figure(figsize=(20,8))
plt.style.use("fivethirtyeight")
plt.plot(range(1, 31), sse)
plt.xticks(range(1, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Elbow method")
plt.show()
```




```
## Find the elbow point
kl = KneeLocator(
    range(1, 31), sse, curve="convex", direction="decreasing")
print("Optimal number of clusters based on elbow method: ", kl.elbow)
```

```
Optimal number of clusters based on elbow method:  4
```

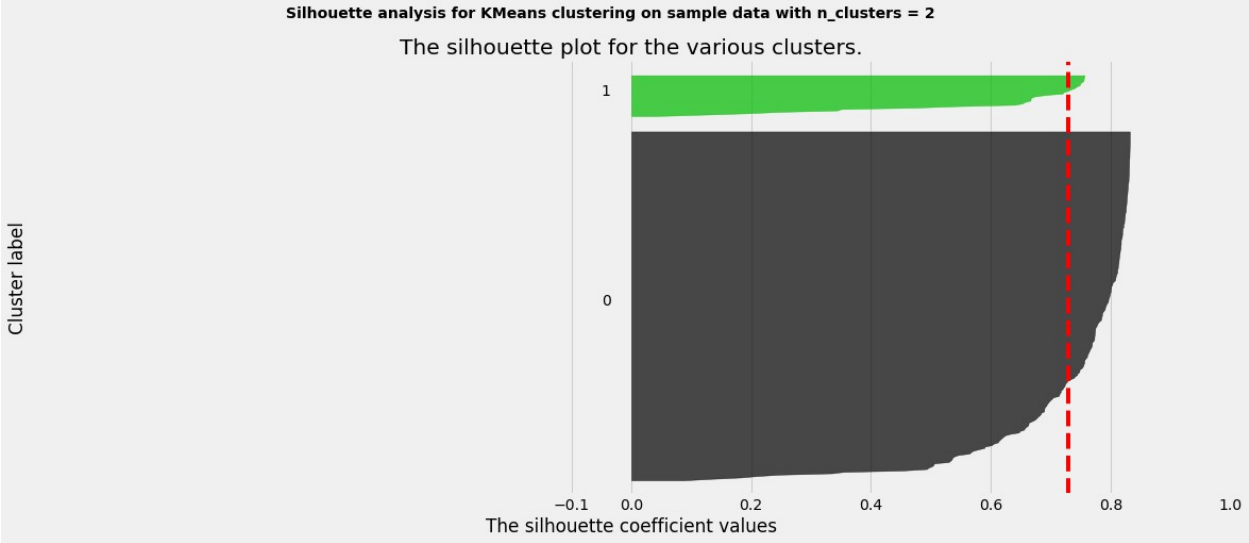
```
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []
for k in range(2, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    score = silhouette_score(scaled_features, clusterer.labels_)
    silhouette_coefficients.append(score)

plt.figure(figsize=(15,6))
plt.style.use("fivethirtyeight")
plt.plot(range(2, 31), silhouette_coefficients)
plt.xticks(range(2, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.title("Silhouette method")
plt.show()
```




```
# let's take a look at what silhouette analysis looks like:
X1 = scaled_features
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters , random_state=55)
kmeans.fit(scaled_features)
cluster_labels = kmeans.labels_
score = silhouette_score(scaled_features, cluster_labels)

fig, ax1 = plt.subplots()
fig.set_size_inches(18, 7)

# The 1st subplot is the silhouette plot
# The silhouette coefficient can range from -1, 1 but in this example
all
# lie within [-0.1, 1]
ax1.set_xlim([-1, 1])
# The (n_clusters+1)*10 is for inserting blank space between
silhouette
# plots of individual clusters, to demarcate them clearly.
ax1.set_ylim([0, len(scaled_features) + (n_clusters + 1) * 10])


# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the
formed
# clusters
silhouette_avg = silhouette_score(scaled_features, cluster_labels)
print("For n_clusters =", n_clusters,
    "The average silhouette_score is :", silhouette_avg)

# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(scaled_features,
cluster_labels)
```

```
y_lower = 10

for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the
middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1,0, 0.2, 0.4, 0.6, 0.8, 1])

plt.suptitle(("Silhouette analysis for KMeans clustering on sample
data "
              "with n_clusters = %d" % n_clusters),
              fontsize=14, fontweight='bold')

plt.show()
```

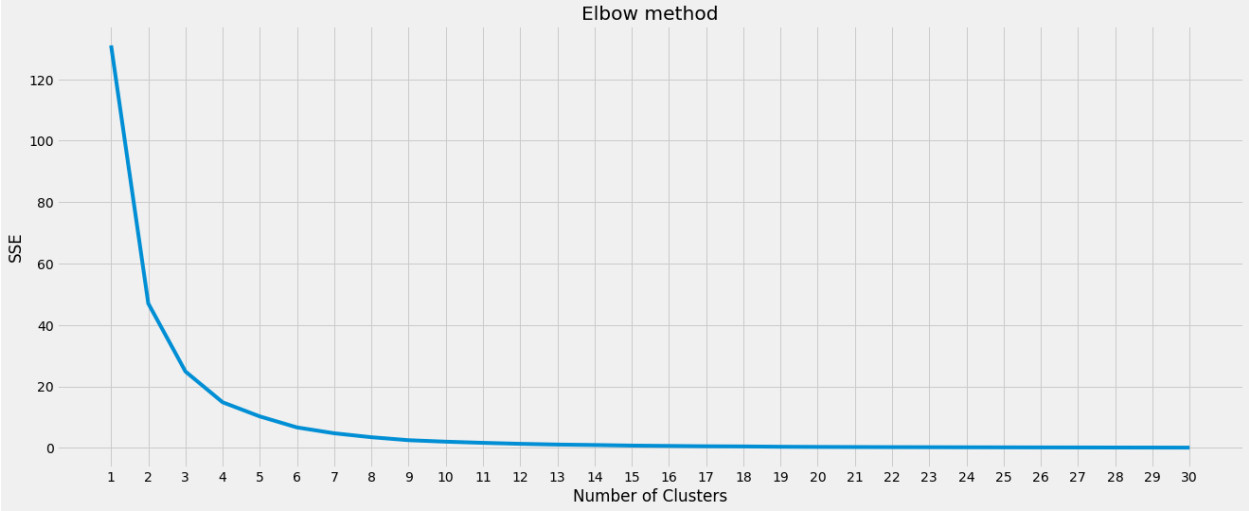
```
For n_clusters = 2 The average silhouette_score is :
0.7288654948840406
```




```
label_df =
pd.DataFrame(cluster_labels,columns=['kmeans_cluster_label'])
result = pd.concat([data,label_df],axis=1)
result.to_csv('kmeans_result_Berk_GB.csv',index=False)
```

The above block saves the labels as a `.csv` file and we see that the last column shows to which cluster they have been put in.
Now we will be working with `All the GB data for Spherical Nano-indentation testing`.

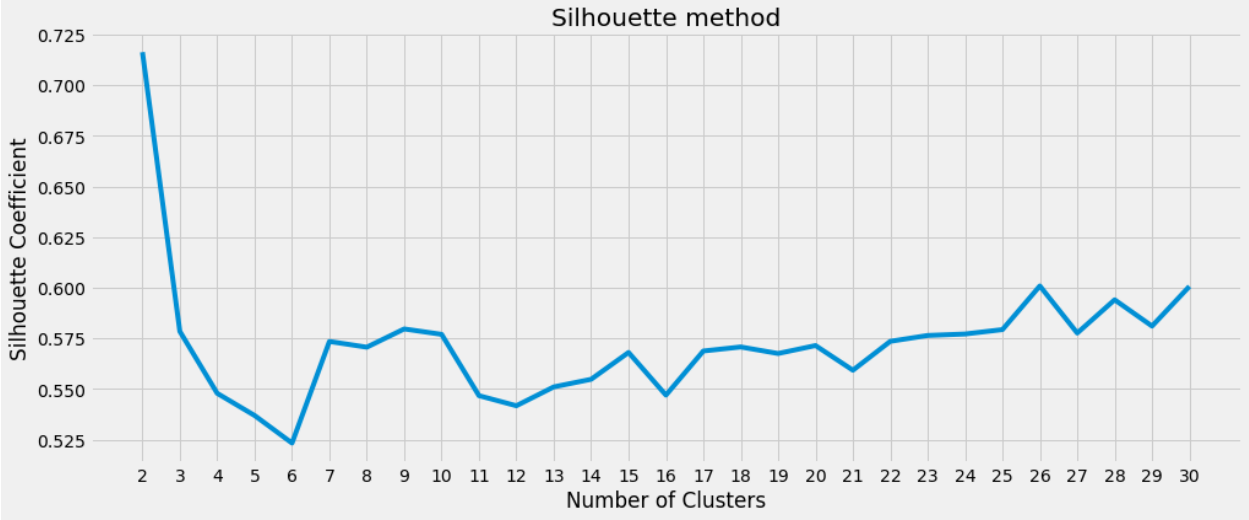
```
data = pd.read_csv("./Hardness_Modulus_spherical_all_GB.csv")
subset = data[['Hardness (Gpa)']]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(subset) #this will scale the
data for each feature that the means are 0, and variances are 1.

# A list holds the SSE values for each k
sse = []
for k in range(1, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    sse.append(clusterer.inertia_)

plt.figure(figsize=(20,8))
plt.style.use("fivethirtyeight")
plt.plot(range(1, 31), sse)
plt.xticks(range(1, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Elbow method")
plt.show()
```




```
## Find the elbow point
kl = KneeLocator(
    range(1, 31), sse, curve="convex", direction="decreasing")
print("Optimal number of clusters based on elbow method: ", kl.elbow)
```

```
Optimal number of clusters based on elbow method:  5
```

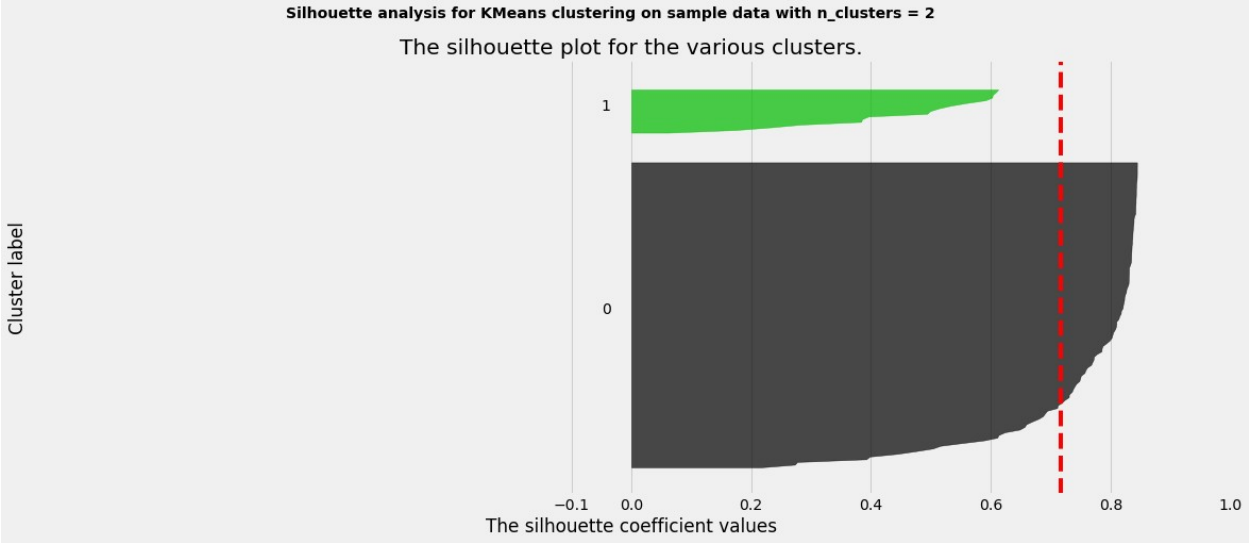
```
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []
for k in range(2, 31):
    clusterer = KMeans(n_clusters=k, random_state=55)
    clusterer.fit(scaled_features)
    score = silhouette_score(scaled_features, clusterer.labels_)
    silhouette_coefficients.append(score)

plt.figure(figsize=(15,6))
plt.style.use("fivethirtyeight")
plt.plot(range(2, 31), silhouette_coefficients)
plt.xticks(range(2, 31))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.title("Silhouette method")
plt.show()
```




```
# let's take a look at what silhouette analysis looks like:
X1 = scaled_features
n_clusters = 2
kmeans = KMeans(n_clusters=n_clusters , random_state=55)
kmeans.fit(scaled_features)
cluster_labels = kmeans.labels_
score = silhouette_score(scaled_features, cluster_labels)

fig, ax1 = plt.subplots()
fig.set_size_inches(18, 7)

# The 1st subplot is the silhouette plot
# The silhouette coefficient can range from -1, 1 but in this example
all
# lie within [-0.1, 1]
ax1.set_xlim([-1, 1])
# The (n_clusters+1)*10 is for inserting blank space between
silhouette
# plots of individual clusters, to demarcate them clearly.
ax1.set_ylim([0, len(scaled_features) + (n_clusters + 1) * 10])


# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the
formed
# clusters
silhouette_avg = silhouette_score(scaled_features, cluster_labels)
print("For n_clusters =", n_clusters,
    "The average silhouette_score is :", silhouette_avg)

# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(scaled_features,
cluster_labels)
```

```
y_lower = 10

for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the
middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1,0, 0.2, 0.4, 0.6, 0.8, 1])

plt.suptitle(("Silhouette analysis for KMeans clustering on sample
data "
              "with n_clusters = %d" % n_clusters),
              fontsize=14, fontweight='bold')

plt.show()
```

```
For n_clusters = 2 The average silhouette_score is :
0.7162864314426384
```




```
label_df =
pd.DataFrame(cluster_labels,columns=['kmeans_cluster_label'])
result = pd.concat([data,label_df],axis=1)
result.to_csv('kmeans_result_Spherical_GB.csv',index=False)
```